www.clubedelphi.net **Feita para desenvolvedores**

ClubeDelphi

DevMedia GROUP

Ano 7 · Edição 91 · R$9,90

ISSN 1517990-7

Agora com + páginas e com duas grandes novidades!

- Artigos de PHP
- e muito mais artigos para iniciantes em Delphi

Introdução ao Firebird 2.0, IBExpert e SQL

Na Web

Confira no portal ClubeDelphi duas vídeo-aulas que ensinam como acessar o SQL Server com o Delphi

Mini-curso
Construa um sistema completo de Contas a Pagar e Cobrança com dbExpress e Firebird 2.0 – Parte 5
Impressão de boletos bancários

Mão na Massa
Aprenda uma maneira simples de captura de dados atualizados na Web

Coluna Ask the Expert

PHP
Mini-curso de Delphi for PHP –
Crie uma aplicação Web completa – Parte 4

PHP
Mini-Curso - Introdução à linguagem PHP – Parte 3

POO e Client/Server
Saiba como colocar em prática o uso dos principais recursos da POO em aplicações reais

Easy Delphi
Introdução à POO - Primeiros passos para entender os principais conceitos de Programação Orientada a Objetos

Banco de Dados
Dicas de dbExpress

Easy Delphi
Treinamento em ASP.NET – Parte 2

Quantas cópias de seu software existem no mercado?

Esta é a chave mais segura do mundo contra pirataria de software.

Comprove você mesmo!

SafeNet®
The Foundation of Information Security

Alameda Tocantins, 280 - Barueri-SP
Tel: +55 11 4208-7700
http://br.safenet-inc.com - Contato: infobrasil@safenet-inc.com

ClubeDelphi

Ano 7 - 91ª Edição - 2007 - ISSN 1517990-7 Impresso no Brasil

Corpo Editorial

Editor Geral
Guinther Pauli
guinther@devmedia.com.br

Editor Técnico
Adriano Santos
adrianosantos@devmedia.com.br

Equipe Editorial
Adriano Santos, Bruno Lichot, Everson Volaco, Fabrício Desbessel, Gustavo Chaurais e Michael Benford

Editor de Arte
Vinicius O. Andrade
viniciusoandrade@gmail.com

Capa
Antonio Xavier
antonioxavier@devmedia.com.br

Revisão
Gregory Monteiro
gregory@clubedelphi.net

Distribuição
Fernando Chinaglia Dist. S/A
Rua Teodoro da Silva, 907
Grajaú - RJ - 206563-900

Atendimento ao Leitor

A DevMedia conta com um departamento exclusivo para o atendimento ao leitor. Se você tiver algum problema no recebimento do seu exemplar ou precisar de algum esclarecimento sobre assinaturas, exemplares anteriores, endereço de bancas de jornal, entre outros, entre em contato com:

Carmelita Mulin
www.devmedia.com.br/central/default.asp
(21) 2215-0033

Kaline Dolabella
Gerente de Marketing e Atendimento
kalined@terra.com.br
(21) 2215-0033

Publicidade

Para informações sobre veiculação de anúncio na revista ou no site entre em contato com:

Kaline Dolabella
publicidade@devmedia.com.br

ClubeDelphi PLUS

DevMedia GROUP

A revista ClubeDelphi é parte integrante da assinatura ClubeDelphi PLUS. Para mais informações sobre o pacote PLUS, acesse: *http://www.devmedia.com.br/clubedelphi/portal.asp*

Apoio

ASSESPRO PARANÁ

SEÇÃO DELPHI: Artigos intermediários sobre Delphi Win32 e Delphi .net.

06 - Contas a Pagar e Cobrança - Parte 5
Adriano Santos

12 - Dicas de dbExpress
Maikel Marcelo Scheid

17 - Trabalhando com informações de páginas HTML
Luciano César Ascari

20 - POO e Client/Server
Maikel Marcelo Scheid

26 - Dicas de RAVE Reports
Maikel Marcelo Scheid

SEÇÃO EASY DELPHI: Artigos para iniciantes em Delphi

32 - Introdução ao Firebird e IBExpert
Maikel Marcelo Scheid

40 - Introdução à POO
Adriano Santos

45 - Treinamento em ASP.NET – Parte 2
Renato Haddad

SEÇÃO PHP: Artigos sobre a linguagem PHP e a ferramenta Delphi for PHP

52 - Delphi for PHP - Parte 4
Fabrício Desbessel

60 - Introdução ao PHP – Parte 3
Ewald Geschwinde e Hans-Juergen Schoenig

EDITORIAL

Seguindo nossa linha de artigos voltados mais para o público iniciante, hoje nosso destaque vai para o banco de dados mais utilizado pelos desenvolvedores Delphi. Se você está iniciando o desenvolvimento de aplicações client/server, mais cedo ou mais tarde vai se confrontar com o banco open source. E que tal dar os primeiros passos com o banco de dados? Aprenda a instalar o Firebird 2.0, configurar o servidor e a utilizar a poderosa ferramenta IBExpert, destinada a manipular BDs, executar comandos SQL etc. No final, tenha uma visão geral da linguagem SQL, utilizada para consulta e manipulação de dados em um BD.

O Adriano Santos continua o mini-curso que mostra como construir uma aplicação completa de contas a pagar e cobrança, utilizando as tecnologias mais comuns do Delphi: Client/Server, dbExpress, Win32 e VCL. Aprenda nesta edição a imprimir boletos bancários utilizando uma solução totalmente free, em alguns minutos!

O Maikel Scheid apresenta dicas de dbExpress, como por exemplo, para criar relacionamentos Master/Detail entre duas tabelas que são tratadas e manipuladas como um único DataPacket. Aprenda também a monitorar a troca de mensagens que acontece entre sua aplicação Delphi e o banco de dados Firebird, através do SQLMonitor. Importante para detectar problemas de performance, por exemplo. E finalmente, veja como utilizar um conjunto reduzido de componentes dbExpress para manipular o banco de dados. Em seu outro artigo, Maikel mostra um pouco sobre POO no mundo Client/Server.

O Luciano apresenta um interessante recurso através dos componentes da paleta Indy: como extrair informações de páginas Web para serem utilizadas em seus aplicativos Delphi. Imagine o leque de possibilidades que isso abre, visto que podemos agregar ao nosso aplicativo praticamente qualquer serviço que seja disponibilizado na Web.

Na seção Easy Delphi, aprenda os conceitos básicos da POO, no artigo de Adriano Santos. Através de exemplos simples, aprenda na prática a utilizar esse paradigma que é sem dúvida um dos pilares de todos os aplicativos que você vier a desenvolver com o Delphi. Ainda na sessão Easy Delphi, temos a continuação do Treinamento em ASP.NET.

E temos muito PHP também. O Fabrício continua a criação do seu site de notícias, mostrando inclusive como utilizar o recurso de Templates. E temos também a continuação do mini-curso que está desvendando todos os segredos desta linguagem que agora já faz parte do universo Delphi.

Despeço-me desejando um Feliz Natal e Próspero Ano Novo a todos os nossos leitores, colaboradores, editores e demais pessoas que contribuem para que a ClubeDelphi seja um sucesso.

Sucesso a todos e boa leitura!

Guinther Pauli
guinther@devmedia.com.br
Microsoft Certified: MCP, MCAD, MCSD.NET
Borland Certified: Delphi 6, 7, 2005, 2006, Web, Kylix

Fale com o Editor

É muito importante para a equipe saber o que você está achando da revista: que tipo de artigo você gostaria de ler, que artigo você mais gostou e qual artigo você menos gostou. Fique a vontade para entrar em contato com os editores e dar a sua sugestão!
Se você estiver interessado em publicar um artigo na revista ou no site ClubeDelphi, entre em contato com os editores, informando o título e mini-resumo do tema que você gostaria de publicar:

Guinther Pauli - Editor da Revista
guinther@devmedia.com.br

Luciano Pimenta - Editor do Site
lucianopimenta@devmedia.com.br

Portal do Assinante

NÃO PERCA

A ClubeDelphi tem uma novidade para você que comprou este exemplar na banca de jornal: você pode acessar GRATUITAMENTE, o Portal do Assinante ClubeDelphi!

Confira o que você encontra no Portal do Assinante:
- Mais de 480 Vídeo Aulas!
- 6 cursos online!
- 1 Livro Eletrônico sobre ADO.NET e BDP!
- Mais de 150 Artigos Exclusivos!

Para Utilizar o Portal do Assinante, acesse *www.devmedia.com.br/clubedelphi/potal.asp* e utilize as informações abaixo: **Login: DVM.CD e Senha: CSJDW**

O acesso é válido por 30 dias a partida da data de lançamento da revista. Todos os meses a ClubeDelphi lhe dará uma senha válida para acessar o portal. Comprando a revista regularmente em bancas, você terá acesso ininterrupto a ele!

Ask The Expert

Perguntas e Respostas

Dúvidas respondidas por **Luciano Pimenta**
(envie as suas para lucianopimenta@devmedia.com.br)

Como criar uma rotina semelhante ao auto-completar do Internet Explorer?

Gostaria de montar uma rotina capaz de auto-completar o texto do *Edit* semelhante ao que acontece quando digitamos um site na barra de endereços do Internet Explorer.

Flávio de Marques

Olá Flávio, respondendo a sua questão vamos criar uma pequena função capaz de localizar em um *ListBox* parte do nome digitado em um *Edit* e sugerir o restante do nome como auto-completar.

Insira um *Edit* e um *ListBox* no formulário. Adicione alguns nomes de pessoas no *ListBox* para que possamos testar. Na seção *public* do formulário declare a seguinte *function:*

```
function Procura(Txt: string): string;
```

Pressione *Ctrl* + *Shift* + *C* para que o Delphi crie pra nós o cabeçalho da função. Digite o código da **Listagem 1**. O código é responsável por localizar, dentro do *ListBox*, o texto digitado no *Edit*.

Em seguida devemos preparar o *Edit* para que possa interagir com a função, para isso digite o código da **Listagem 2** no evento *OnChange* e o código da **Listagem 3** em *OnKeyDown*.

Veja na **Figura 1**, como ficou o programa em execução.

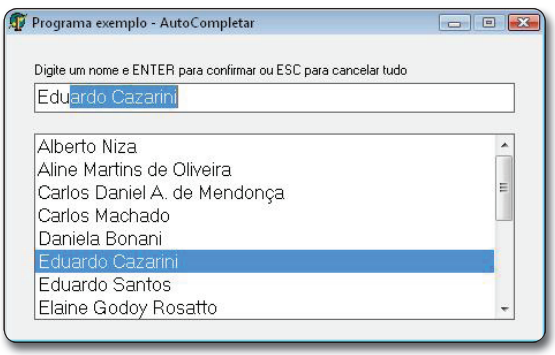


Figura 1. Exemplo auto-completar

Listagem 1. Código da função Procura

```
function TForm1.Procura(Txt: string): string;
var
  I : Integer;
begin
  with ListBox1.Items do
    if Txt <> '' then
      for I := 0 to Pred(Count) do
        if (LSTRCMP(Pchar(Copy(Strings[I], 1,
          Length(Txt))), Pchar(TXT)) = 0) or
          (LSTRCMPI(Pchar(Copy(Strings[I], 1,
          Length(Txt))), Pchar(Txt)) = 0) then
        begin
          Result := Strings[I];
          Break;
        end;
end;
```

Listagem 2. Evento Change do Edit

```
procedure TForm1.Edit1Change(Sender: TObject);
var
  I, J : Integer;
begin
  if not bPausa and Active then
  begin
    if Procura(Edit1.Text) <> '' then
    begin
      J := Length(Edit1.Text);
      Edit1.Text := Procura(Edit1.Text);
      Edit1.SelStart := J;
      Edit1.SelLength:= Length(Procura(Edit1.Text))-J;
    end;
  end;
  bPausa := False;
  for I := 0 to ListBox1.Items.Count - 1 do
    if ListBox1.Items.Strings[I] = Edit1.Text then
    begin
      ListBox1.ItemIndex := I;
      Break;
    end
    else
      ListBox1.ItemIndex := -1;
end;
```

Listagem 3. Código do evento KeyDown do Edit

```
procedure TForm1.Edit1KeyDown(Sender: TObject;
  var Key: Word; Shift: TShiftState);
begin
  if (Key = VK_BACK) or (Key = VK_DELETE) then
    bPausa := True;
  if Key = 13 then
    Edit1.SelStart := Length(Edit1.Text);
  if Key = 27 then
    Edit1.Clear;
end;
```

Assinatura
ClubeDelphi PLUS
Mais conteúdo .NET por menos!

+480 vídeo aulas | 6 cursos online

www.clubedelphi.net/portal

Caro Leitor,

O portal ClubeDelphi PLUS é a continuação, na Web, da revista ClubeDelphi. O portal recebe um conteúdo novo todo dia e hoje conta com: i) mais de 480 vídeo aulas; ii) 6 cursos online; iii) 1 livro eletrônico gratuito, de Guinther Pauli, sobre ADO.NET e BDP; iv) mais de 150 artigos exclusivos (que não foram publicados na revista)!;

Acesse o portal ClubeDelphi PLUS e receba muito mais conteúdo sobre Delphi! E o que é melhor: de graça! Todo leitor da revista ClubeDelphi, seja ele assinante ou comprador da revista em bancas, tem acesso ao portal (para quem compra em bancas, o acesso é válido por 30 dias).

Se você é assinante, utilize o seu login e senha pessoais para acessar o portal. Se você comprou em bancas, utilize o login e senha publicados na página do editorial desta edição.

Confira a seguir as últimas novidades do portal!

Boa leitura e sucesso!
Equipe DevMedia

.:: DESTAQUE ::.

Confira no portal ClubeDelphi PLUS 2 vídeo aulas sobre dbGo com Delphi 2007

Acessando SQL Server com dbGo no Delphi 2007
http://www.devmedia.com.br/articles/viewcomp.asp?comp=7256
http://www.devmedia.com.br/articles/viewcomp.asp?comp=7349

Curso Online - PLUS

O portal ClubeDelphi PLUS conta com 6 cursos online, confira!

Sistema completo com Delphi 7, dbExpress e Firebird 2.0 [*Luciano Pimenta*]

Aplicações WEB com IW e Delphi 7 (Delphi Win32)! [*Guinther Pauli*]

Aplicações client/server no Delphi 2006 [*Luciano Pimenta*]

Criando uma Aplicação multi-camadas Completa com Delphi [*Guinther Pauli*]

Criando uma aplicação completa: sistema SysCash [*Everson Volaco*]

Aplicações Client/Server com dbExpress e Firebird [*Luciano Pimenta*]

Últimas Vídeo-Aulas (Somente para assinantes)

Aprenda a desenvolver sistemas para o sistema operacional PalmOS
Acompanhe as aulas de Ricardo Boaro que falam unicamente do desenvolvimento de aplicações para PalmOS utilizando a IDE PocketStudio que é bastante semelhate ao Delphi inclusive utilizando-se de linguagem Pascal.

Desenvolvendo uma aplicação para PalmOS com PocketStudio - Parte VII
Veja na vídeo aula de Ricardo Boaro, como trabalhar com aplicações Palm/OS com o PocketStudio.

Acessando SQL Server com dbGo no Delphi 2007
Veja nesta vídeo-aula de Ricardo Boaro como acessar um banco SQL Server usando dbGo no Delphi 2007.

Imprimindo etiquetas no QuickReport - Parte IV a V
Veja nessas vídeo-aulas de Paulo Quicoli como criar etiquetas personalizadas usando o editor de relatórios nativo do Delphi, QuickReport.

Trabalhando com Diretivas de Compilação
Veja nessa vídeo aula de Ricardo Boaro, como fazer uso de diretivas de compilação no Delphi.

Introdução ao componete WPTools
Veja nessa vídeo aula de Paulo Quicoli, como trabalhar com o componente WPTools no Delphi. Esse componente tem como principal finalidade possibilitar a edição de textos do Word dentro da própria aplicação.

Nesta seção você encontra artigos intermediários sobre Delphi Win32 e Delphi .NET

Contas a Pagar e Cobrança

Crie um sistema completo com Delphi, Firebird 2.0 e dbExpress - Parte 5

Adriano Santos
(falecom@adrianosantos.pro.br)
é desenvolvedor Delphi desde 1998. Professor e programador PHP. Bacharel em Comunicação Social pela Universidade Cruzeiro do Sul, SP. É Editor Técnico, Colunista e Membro da Comissão Editorial da revista ClubeDelphi. Mantém o blog Delphi to Delphi (www.delphitodelphi.blogspot.com) com dicas, informações e tudo sobre desenvolvimento Delphi.

Nosso mini-curso *SysPague* - Sistema de Contas a Pagar e Cobrança está chegando ao fim nesta edição. Durante as quatro últimas edições da revista ClubeDelphi vimos como criar um sistema de gerenciamento de contas a pagar e cobrança, desde a criação dos principais cadastros até a exportação dos dados para envio ao banco e, conseqüentemente, a geração de boletos bancários.

Nessa quinta e última parte do curso veremos como criar nossos próprios boletos bancários utilizando um componente gratuito no Delphi.

Para iniciar nosso artigo, veremos rapidamente o que foi criado no exemplo anterior:

- Criação de *Stored Procedure* para seleção dos boletos;
- Desenvolvimento da tela de *Cobrança Escritural*;

Como mencionado anteriormente, faremos o desenvolvimento dos boletos bancários diretamente pelo sistema, dando maior flexibilidade ao usuário final.

Usaremos o componente *Delphi Boleto* desenvolvido pelo colega Genilton Barbosa, que mantém um grupo de discussão onde encontramos não somente o componente, mas também dicas, exemplos e diversas versões do componente para os mais variados editores de relatório.

Veremos onde efetuar o *download*, como instalar e manusear o componente para geração dos boletos.

Fazendo download e instalação

A primeira parte deste artigo consiste em efetuar o *download* do componente e então fazer sua instalação na *IDE*. Como de costume, usaremos a versão 7.0 do Delphi.

Os arquivos de instalação do *Delphi Boleto* ficam hospedados na seção *Arquivos* do grupo criado pelo autor no site do Yahoo! Grupos. Acesse o endereço do grupo pelo link *br.groups.yahoo.com/group/delphi-boleto*, clique em *Entre neste grupo!* e

preencha os dados requeridos. Após confirmar, aguarde até que os moderadores aceitem sua inscrição.

Após ser aceito no grupo, entre novamente no link indicado e veja que os itens de menu à esquerda do site encontram-se disponíveis. Clique em *Arquivos* onde encontrará diversos outros links e pastas. Localize o link *Delphi Boleto.zip* para Delphi 7.0. Baixe-o e descompacte-o em uma pasta de sua preferência.

Descompactados os arquivos basta efetuar a instalação dos mesmos no *IDE*. Para isso, abra o Delphi, feche todos os projetos ativos e então clique em *File>Open*. Localize o arquivo *gCob7.dpk*, compile o projeto e instale.

Note que é criada uma nova paleta de componentes na *IDE* com o nome *GBSoft*, e nela dois componentes foram adicionados conforme a **Figura 1**.

Preparando o banco de dados

No artigo anterior criamos uma *SP(Stored Procedure)* selecionável capaz de receber um *range* de datas, filtrar e selecionar os dados para geração da *Cobrança Escritural*.

Nessa quinta parte usaremos a mesma técnica empregada anteriormente para que possamos resgatar os registros necessários para então gerarmos os boletos bancários. Usaremos a mesma *SP*, porém desta vez seus dados de retorno serão usados para alimentar o componente *Delphi Boleto*. A **Listagem 1** mostra o script de criação da *SP* para que possamos nos lembrar de quais dados ela nos retorna.

Note que os parâmetros de saída são: *Codigo, CNPJ, Valor_Real, Dt_Cadastro, Dt_Vecto, Razão, Endereço, Bairro, Cidade, Estado* e *CEP*. De todos os parâmetros o único que não usaremos é o *Dt_Cadastro*. O esquema é o mesmo do artigo anterior e poderemos usar inclusive o mesmo *ClientDataSet* para retornar as informações necessárias.

Alterando o banco de dados

Precisaremos fazer uma pequena alteração em nosso banco de dados para receber uma nova informação na tabela de *Contas Bancárias* do sistema: o *Cedente*.

O código *Cedente* é o que identifica a conta da empresa internamente no banco. Muitos bancos utilizam como código *Cedente* o número da agência concatenado ao número da conta corrente. Outros utilizam um número completamente diferente, muitas vezes o contrato da empresa junto à instituição bancária. Sendo assim é altamente recomendado solicitar ao seu cliente qual o número de *Cedente* que ele recebeu do banco ao contratar o serviço de cobrança bancária, para então alimentar corretamente o sistema. Esse número varia de 8 a 20 caracteres em média, portanto criaremos um campo *Varchar* de 20 posições.

Abra o IBExpert ou o seu gerenciador de bancos de dados preferido e insira um novo campo na tabela *Contas* com o nome *Cedente* do tipo *Varchar* e tamanho igual a 20. Se preferir rode o *script* a seguir para fazer a alteração:

```
ALTER TABLE CONTAS ADD CEDENTE VARCHAR(20)
```

Feito isso será necessário fazer a inclusão do novo campo também no sistema. Acesse o *Data Module* principal, encontre o *sqlContas* e clique duas vezes em sua propriedade *SQL*. Adicione o campo *Cedente* a *Select* criada anteriormente. Nossa instrução *SQL* deverá se parecer com a **Listagem 2**.

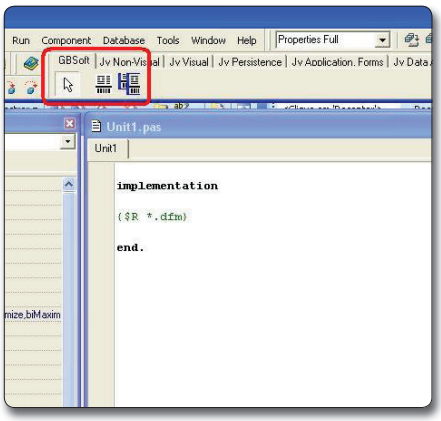


Figura 1. Paleta de componentes GBSoft Delphi Boleto

Listagem 1. Stored Procedure de seleção dos boletos

```
SET TERM ^ ;

CREATE PROCEDURE RETORNARBOLETOS (
    DT_INICIO TIMESTAMP,
    DT_FIM TIMESTAMP)
RETURNS (
    CNPJ VARCHAR(18),
    VALOR_REAL DOUBLE PRECISION,
    DT_CADASTRO TIMESTAMP,
    DT_VECTO TIMESTAMP,
    CODIGO VARCHAR(20),
    RAZAO VARCHAR(150),
    ENDERECO VARCHAR(100),
    BAIRRO VARCHAR(50),
    CIDADE VARCHAR(50),
    ESTADO VARCHAR(2),
    CEP VARCHAR(9))
AS
begin
  FOR SELECT
    CP.CODIGO, CP.CNPJ, CP.VLR_REAL,
    CP.DT_CADASTRO, CP.DT_VECTO, CL.RAZAO,
    CL.ENDERECO, CL.BAIRRO, CL.CIDADE,
    CL.ESTADO, CL.CEP
    FROM
      CONTAS_RECEBER CP
      INNER JOIN CLIENTES CL
      ON (CP.CNPJ = CL.CNPJ)
    WHERE
      (CP.DT_VECTO BETWEEN :DT_INICIO AND :DT_FIM) AND
      (CP.STATUS NOT IN ('I', 'B'))
    INTO
      :CODIGO, :CNPJ, :VALOR_REAL, :DT_CADASTRO,
      :DT_VECTO, :RAZAO, :ENDERECO, :BAIRRO, :CIDADE,
      :ESTADO, :CEP
  DO
    SUSPEND;
end^
```

Listagem 2. Select para retornar os registros da tabela Contas

```
SELECT
  BANCO, AGENCIA, CONTA, NOME_AGENCIA, NOME_CONTA,
  ENDERECO,BAIRRO, COMPLEMENTO, CIDADE, ESTADO, CEP,
  TELEFONE, FAX, DT_CADASTRO, CONTATO, STATUS,
  DT_ALTERACAO, CEDENTE
FROM
  CONTAS
WHERE
  STATUS = 'A'
ORDER BY
  BANCO, AGENCIA, CONTA
```

Após alterar a propriedade descrita anteriormente acesse o *Fields Editor* do componente *sqlContas* através do menu de contexto ou clique duas vezes no componente. Clique novamente com o botão direito e acione o item *Add fiedls*, selecione o campo *Cedente* e confirme.

Acabamos de incluir o campo no *Data Module* principal, agora é necessário fazer sua inclusão na janela de cadastro para que o usuário possa alterar esse dado em tempo de execução. Por isso abra o formulário *frmContas*, inclua um novo *DBEdit*("dbeCedente") da paleta *Data Controls* e ligue-o ao campo adicionado usando as propriedades *Data Source* e *Data Field*. Veja como fica nossa tela de cadastro na **Figura 2.**

Se necessário compile o projeto novamente, execute-o e teste a janela de cadastro no intuito de verificar se o campo adicionado está funcionando corretamente.

Gerando os boletos bancários

Assim como mencionado anteriormente iremos utilizar a mesma *SP* gerada no artigo anterior já que necessitamos dos mesmos valores para geração dos boletos.

Crie um novo formulário através do menu *File|New>Form*. Salve-o como "uBoletos.pas" e mude sua propriedade *Name* para "frmBoletos". Desenhe a tela conforme a **Figura 3** e note que utilizamos o mesmo *box* utilizado na tela *Cobrança Escritural* com apenas uma diferença: o código do *Cedente*.

Listagem 3. Evento do botão sbbSelecionarBanco

```
procedure TfrmBoletos.sbbSelecionarBancoClick(Sender: TObject);
begin
  frmPesquisaPadrao := TfrmPesquisaPadrao.Create(Self);
  with frmPesquisaPadrao do
  begin
    FTabela := 'Contas';
    FDataSetAlvo := dmdPrincipal.cdsContas;
    ShowModal;

    if (ModalResult = mrOk) and not
      (dstGenerico.DataSet.IsEmpty) then
    begin
      edtBanco.Text := dstGenerico.DataSet.FieldByName('BANCO').AsString;
      edtAgencia.Text := dstGenerico.DataSet.FieldByName('AGENCIA').AsString;
      edtConta.Text := dstGenerico.DataSet.FieldByName('CONTA').AsString;
      edtCedente.Text := dstGenerico.DataSet.FieldByName('CEDENTE').AsString;
    end;
  end;
  FreeAndNil(frmPesquisaPadrao);
end;
```

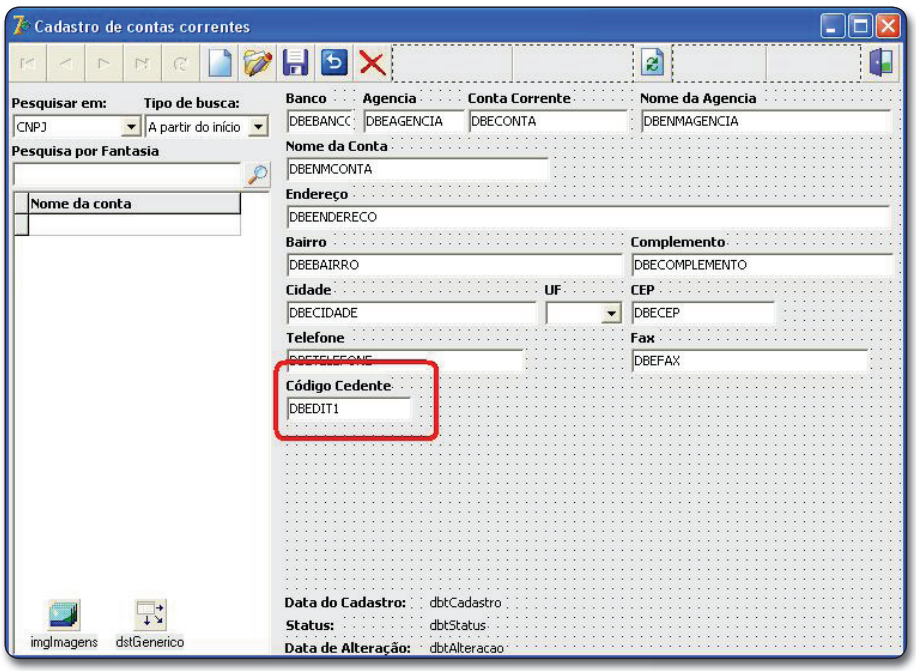


Figura 2. Cadastro de Contas Bancárias alterado

Já que a tela é bem parecida, recomendo que copie completamente os componentes do formulário *frmCobrancaEscritural* e cole-os em *frmBoletos* bastando apenas alterar a largura do formulário e adicionar um novo *Edit*("edtCedente") para receber o código. Marque como *False* a propriedade *Enabled* do *edtCedente* para evitar erros futuros já que esse dado não deve ser variável como os demais.

Nessa janela incluímos também um botão para busca do banco, agência e conta corrente. Ao clicar no botão o *cedente* também será copiado da tabela de *Contas* para a janela atual.

O código para busca é o mesmo criado na caixa de diálogo *Cobrança Escritural*, de qualquer forma podemos ver o código, já comentado na edição anterior, na **Listagem 3** acrescido apenas da linha que copia o conteúdo do campo *Cedente* para a janela atual prevista na linha a seguir:

```
edtCedente.Text :=
  dstGenerico.DataSet.FieldByName('CEDENTE').
    AsString;
```

Outra providência importante é fechar o *cdsBoletos* na abertura da janela para evitar que sejam carregados dados de uma consulta anterior. Por isso insira o código a seguir no evento *OnShow* do formulário:

```
dmdPrincipal.cdsBoletos.Close;
```

> **Nota:** Não esqueça que para o formulário atual "enxergue" os componentes de acesso a dados a *Unit* do *Data Module* principal precisa ser adicionada ao *Uses* do formulário. Para isso será necessário pressionar *ALT + F11* ou ir ao menu *File|Use unit*.

Nesse ponto do artigo estamos quase chegando ao fim. Basta agora adicionarmos um componente do tipo *gbTitulo*("gbBoleto") ao formulário e fazer as devidas codificações. Antes disso vejamos quais as principais propriedades e suas devidas funcionalidades:

- *Cedente*: O *cedente* é nosso cliente, o dono do boleto bancário e conseqüentemente, quem fará o recebimento dos valores pagos pelos boletos;
  - Nesse grupo, o desenvolvedor deve informar obrigatoriamente alguns dados para cobrança tais como: banco, agência bancária e dígito, conta corrente e seu dí-

gito, código do *cedente* (já vimos no início do artigo) e dígito do *cedente*. Ainda se faz necessário Nome, tipo de inscrição (*CPF = tiPessoaFisica, CNPJ = tiPessoaJuridica*) e o número do CPF ou CNPJ de acordo com a necessidade;

• *Sacado*: O sacado é exatamente o contrário de cedente, consiste na parte devedora, aquele que deverá efetuar o pagamento do boleto.

  ○ Aqui devemos apenas informar: endereço, cidade, estado, cep, tipo de inscrição (*CPF = tiPessoaFisica, CNPJ = tiPessoaJuridica*) e o número do CPF ou CNPJ de acordo com a necessidade;

• *Carteira*: Essa informação foi comentada no artigo anterior, consiste no tipo de cobrança contratada por nosso cliente junto à instituição bancária. Nesse exemplo estou usando a carteira *109*. Verifique junto ao banco e/ou cliente qual a carteira deverá ser utilizada. Essa informação também poderia estar previamente configurada na tabela *Contas* bastando apenas ter um campo para isso como fizemos com o *Cedente*;

• *DataVencimento*: Evidentemente o vencimento do boleto bancário;

• *Instrucoes*: Essa propriedade serve para que o desenvolvedor possa incluir as instruções para protesto, tais como: *"Protestar após 30 dias", "Não receber após o vencimento"* etc.;

• *NossoNumero*: O *NossoNumero* é o número do boleto bancário que deve ser único; No exemplo estou colocando como nosso número o código da conta a receber. Esse código é livre e pode ser criado pelo desenvolvedor;

• *ValorDocumento*: Valor real do documento, ou seja, qual o valor do boleto;

Existem, claro, outros campos que devem ser utilizados, tais como: *ValorDesconto, ValorDespesaCobranca, DataProtesto, LocalPagamento* cada qual com sua finalidade.

Você deve estar se perguntando o porquê de existir tantas propriedades se usamos apenas algumas. Bem, o componente *Delphi Boleto* pode ser usado também na geração da *Cobrança Escritural*. Isso mesmo! Esse componente em conjunto com o *gbCobranca* geram os arquivos texto para envio à instituição financeira. Não fiz uso dele no artigo anterior justamente para mostrar como é fácil fazer a geração do texto mesmo que manualmente. Não há segredo, basta saber o *layout* e fazer o desenvolvimento. A principal vantagem do componente é que está tudo pronto e funcional.

Voltando ao exemplo, agora que já conhecemos e sabemos como funciona o *gbTitulo* finalizaremos o mini-curso fazendo a programação final. Em nosso botão *Confirmar* ("btnGerar") insira o código da **Listagem 4** e vamos às explicações.

O código da **Listagem 4** é bem simples de se entender. Primeiro declaramos em um bloco *With* o *Data Module* principal, nosso *cdsBoletos* e o componente *gbBoletos*, isso para termos mais clareza no código e economizarmos algumas digitações. Logo após alimentamos os dados do *Cedente* utilizando as informações de *banco, agência, conta* e *código cedente* presentes na caixa de diálogo.

> **Nota:** Poderíamos ter resgatado essas informações, com exceção do código *cedente*, do próprio cadastro de *contas a receber*, mas lembre-se: *banco, agência* e *conta bancária* somente são atualizados quando é dada a baixa do documento e nesse caso o boleto deve ser gerado primeiro.

Em seguida alimentamos a *query sqlBoletos* com a data de *Início* e *Fim*, período de vencimento das contas a receber. Isso fará um filtro na tabela *Contas_Receber* através da *SP*("RetornarBoletos") que nos trará todos os registros.

O próximo passo é verificar se foram

Figura 3. Caixa de diálogo de geração dos boletos bancários

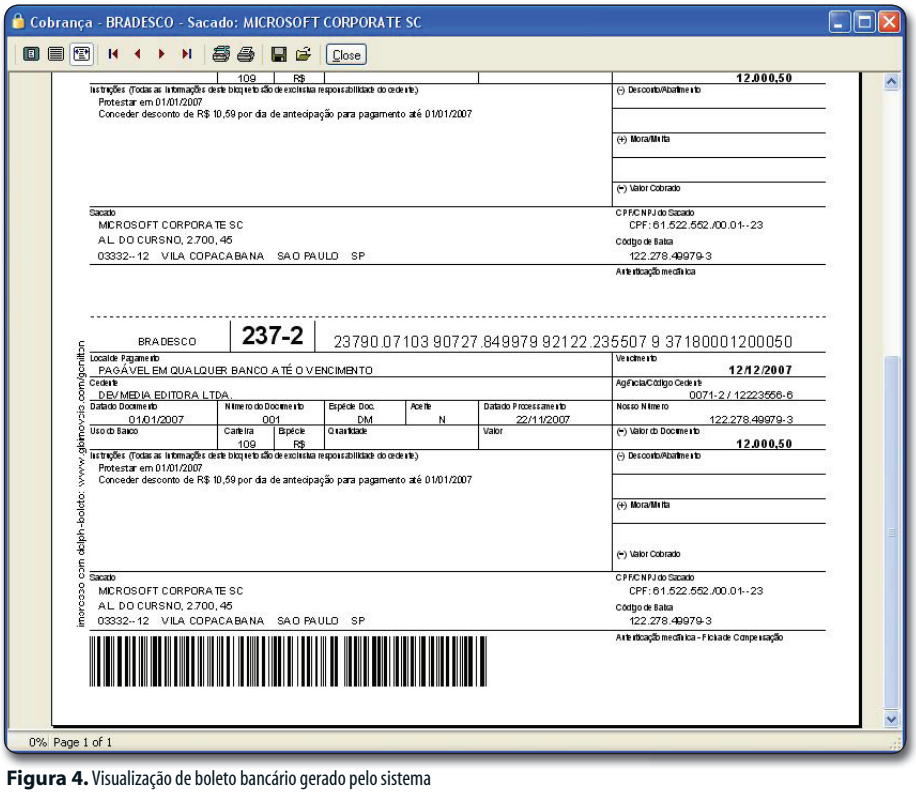


Figura 4. Visualização de boleto bancário gerado pelo sistema

Listagem 4. Código de geração dos boletos (btnGerar)

```
procedure TfrmBoletos.btnGerarClick(Sender: TObject);
begin
  with dmdPrincipal, cdsBoletos, gbBoleto do
  begin
    { Dados Bancários do Cedente }
    Cedente.CodigoCedente := SoNumeros(edtCedente.Text);
    Cedente.DigitoCodigoCedente := SoNumeros(Copy(edtCedente.Text, Length(edtCedente.Text), 1));
    Cedente.ContaBancaria.Banco.Codigo := SoNumeros(edtBanco.Text);
    Cedente.ContaBancaria.CodigoAgencia := Copy(SoNumeros(edtAgencia.Text), 1,
      Length(SoNumeros(edtAgencia.Text))-1);
    Cedente.ContaBancaria.DigitoAgencia := SoNumeros(Copy(edtAgencia.Text,
      Length(edtAgencia.Text), 1));
    Cedente.ContaBancaria.NumeroConta := Copy(SoNumeros(edtConta.Text), 1,
      Length(SoNumeros(edtConta.Text))-1);
    Cedente.ContaBancaria.DigitoConta := SoNumeros(Copy(edtConta.Text, Length(edtConta.Text), 1));

    { Dados de endereço do Cedente }
    Cedente.Endereco.Bairro := 'Jardim das Oliveiras';
    Cedente.Endereco.CEP := '03440890';
    Cedente.Endereco.Cidade := 'São Paulo';
    Cedente.Endereco.EMail := 'cobranca@devmedia.com.br';
    Cedente.Endereco.Estado := 'SP';
    Cedente.Endereco.Numero := '300';
    Cedente.Endereco.Rua := 'Rua Ramos Doces';
    Cedente.Nome := 'Devmedia Editora Ltda.';
    Cedente.NumeroCPFCGC := '60.890.889/0001-10';
    Cedente.TipoInscricao := tiPessoaJuridica;

    { Dados do Sacado }
    FetchParams;
    Params.ParamByName('DT_INICIO').AsSQLTimeStamp := StrToSQLTimeStamp(DateTimeToStr(
        dbeDtInicio.DateTime) + ' 00:00:01');
    Params.ParamByName('DT_FIM').AsSQLTimeStamp := StrToSQLTimeStamp(DateTimeToStr(
        dbeDtFim.DateTime) + ' 23:59:59');
    Open;

    if not IsEmpty then
    begin

      while not EOF do
      begin
        Sacado.ContaBancaria.Banco.Codigo := SoNumeros(edtBanco.Text);
        Sacado.ContaBancaria.CodigoAgencia := Copy(SoNumeros(edtAgencia.Text), 1,
          Length(SoNumeros(edtAgencia.Text))-1);
        Sacado.ContaBancaria.DigitoAgencia := SoNumeros(Copy(edtAgencia.Text,
          Length(edtAgencia.Text), 1));
        Sacado.ContaBancaria.NumeroConta := Copy(SoNumeros(edtConta.Text), 1,
          Length(SoNumeros(edtConta.Text))-1);
        Sacado.ContaBancaria.DigitoConta := SoNumeros(Copy(edtConta.Text, Length(edtConta.Text), 1));
        Sacado.Nome := FieldByName('RAZAO').AsString;
        Sacado.NumeroCPFCGC := FieldByName('CNPJ').AsString;
        Sacado.Endereco.Rua := FieldByName('ENDERECO').AsString;
        Sacado.Endereco.Bairro := FieldByName('BAIRRO').AsString;
        Sacado.Endereco.Cidade := FieldByName('CIDADE').AsString;
        Sacado.Endereco.Estado := FieldByName('ESTADO').AsString;
        Sacado.Endereco.CEP := FieldByName('CEP').AsString;
        DataVencimento := FieldByName('DT_VECTO').AsDateTime;
        NossoNumero := FieldByName('CODIGO').AsString;
        ValorDocumento := FieldByName('VALOR_REAL').AsFloat;
        gbBoleto.Visualizar;
        Next;
      end;
    end
    else
      MessageDlg('Nada selecionado para este período!', mtWarning, [mbOk], 0);
  end;
end;
```

Listagem 5. Código de chamada da tela de boletos

```
procedure TfrmPrincipal.EmissodeBoletosBancrios1Click(Sender: TObject);
begin
  frmBoletos := TfrmBoletos.Create(Self);

  try
    frmBoletos.ShowModal;
  finally
    FreeAndNil(frmBoletos);
  end;

end;
```

retornados registros e em caso positivo fazemos um laço *While* gerando os boletos. Cada vez que o laço é disparado, os dados do sacado são atualizados e o boleto é visualizado em tela. O mais correto aqui é fazer a impressão direta para a impressora evitando que o usuário precise intervir para imprimir em papel e fechar a janela de *preview* toda vez. Por isso ao invés de usar o método *Visualizar* use *Imprimir*, como a seguir:

```
gbBoletos.Imprimir;
```

Se perceber verá que a grande maioria das instruções estão sendo envolvidas pela função *SoNumeros*. Essa é a mesma *procedure* criada no artigo da edição 90. Ela foi redeclarada nesse formulário evitando ter que adicionar ao *Uses* do formulário o *frmCobrancaEscritural*. O mais sensato nesse caso é criar uma *Unit* somente de funções e fazer uso dela, evitando re-digitação desnecessária de código. Veja na **Figura 4** a visualização final do sistema.

Não esqueça de adicionar um novo item de menu ao *MainMenu* da janela principal para que a tela possa ser chamada. O código para chamar o formulário pode ser visto na **Listagem 5**.

Conclusão

Finalizamos a quinta e última parte deste mini-curso onde vimos algumas das principais características de um Sistema de Contas a Pagar e Cobrança. É certo que um sistema como este não se resume apenas em cadastros e emissão de boletos bancários. Ao longo da vida o software acaba ganhando novas funcionalidades, tais como relatórios de fluxo de caixa, emissão e impressão de cheques, controle de centros de custos e uma série de outras características.

Recomendo que reveja partes do sistema e refaça alguns dos passos vistos incluindo novas funcionalidades e agregando desta forma mais valor ao exemplo implementando, por exemplo, uma rotina de *Previsão de Saldo*.

Um forte abraço e até a próxima. ●

POR QUE A ALOG
É LÍDER EM HOSTING GERENCIADO?

- Dois Data Centers próprios – RJ e SP
- Projetos de Hosting customizados
- Foco em serviços de Data Center
- Gerência total dos servidores
- 850 clientes corporativos
- 8 mil m² de área construída
- Destaque do Ano no segmento Internet Serviços no Anuário Telecom 2007

EMPRESA CERTIFICADA ISO 9001 ANAB ACCREDITED BSI

EQUIPE CERTIFICADA EM:

PMI	PMP (PROJECT MANAGEMENT PROFESSIONAL)
ITIL	ITIL FOUNDATION
MICROSOFT	MCP (MICROSOFT CERTIFIED PROFESSIONAL), MCDST (MICROSOFT CERTIFIED DESKTOP SUPPORT TECHNICIAN), MCSE (MICROSOFT CERTIFIED SYSTEM ENGINEER)
CISCO	CCNA (CISCO CERTIFIED NETWORK ASSOCIATE) CCDA (CISCO CERTIFIED DESIGN ASSOCIATE)
RED HAT	RHCE (RED HAT CERTIFIED ENGINEER)
LPI INSTITUTE	LPI-1 (LINUX PROFESSIONAL INSTITUTE CERTIFIED LEVEL 1) LPI-2 (LINUX PROFESSIONAL INSTITUTE CERTIFIED LEVEL 2)
GIAC	GSEC (GIAC SECURITY ESSENTIALS GOLD)
CHECKPOINT	CCSA (CHECKPOINT CERTIFIED SECURITY ADMINISTRATOR)

ALOG
DATA CENTERS DO BRASIL

www.alog.com.br | 0800 282 3330

Hosting Gerenciado® | Colocation | Contingência | Email Corporativo | Conectividade | Serviços Profissionais

Nesta seção você encontra artigos intermediários sobre Delphi Win32 e Delphi .NET

Dicas de dbExpress

Conheça os principais conceitos de uso do dbExpress

Maikel Marcelo Scheid
(maikelscheid@gmail.com)
é técnico em Informática com ênfase em Análise e Programação de Sistemas. Atua na área de Desenvolvimento de Softwares em Delphi para plataforma Win32 e .NET com banco de dados Firebird e MSSQL. É membro da Equipe Editorial ClubeDelphi.

Considerados por muitos programadores como "fantásticos", os componentes da paleta *dbExpress* estão entre os mais utilizados em sistemas *Client/Server* com acesso principalmente aos bancos de dados Firebird/Intebase e MSSQL.

Veremos neste artigo, dicas do uso destes componentes, relacionamentos e uso em conjunto com componentes de outras paletas. Vamos criar uma aplicação com acesso ao banco de dados *Employee.fdb* que acompanha a instalação do Firebird e simular um cadastro e consulta utilizando *master/detail*, monitorando os comandos *SQL* que são trocados pela aplicação e pelo banco de dados.

Para finalizar, veremos uma maneira prática de simplificar o uso de componentes de ligação e consulta ao *BD*. Veremos como estruturar vários componentes *ClientDataSet* em diferentes consultas passando todos por um único componente *SQLDataSet*.

Criando a aplicação

Neste artigo utilizarei o Delphi 7 para criação da aplicação e demonstração dos exemplos e dicas de *dbExpress*, mas fique livre para utilizar outra versão do Delphi para criação do projeto na plataforma Win32.

Crie uma nova aplicação através do menu *File|New>Application*. Altere a propriedade *Name* do formulário inicial para "frmPrincipal" e a propriedade *Caption* para "Dicas dbExpress". Salve a *Unit* principal do projeto como "uPrincipal.pas" e o projeto salve como "prjDicas_dbExpress.dpr".

Para criar a conexão com banco de dados e estruturar os componentes, vamos utilizar um *Data Module* que deverá ser criado através do menu *File|New>Data Module*. Altere seu nome para "DM" e salve a *Unit* como "uDM.pas". Adicione um componente *SQLConnection*("sqlConexao") da paleta *dbExpress*.

O *SQLConnection* é o responsável pela

conexão com o arquivo físico do banco de dados, em nosso caso será o arquivo *Employee.fdb*, normalmente situado em *C:\Arquivos de Programas\Firebird\Firebird(versão)\examples*. Por meio dele obtemos todas as informações da base de dados e fazemos a conexão direta com o *BD*.

Para configurar os parâmetros de acesso ao banco de dados, com um duplo clique sobre o componente *sqlConexao*, crie uma nova conexão utilizando o ícone *Add Connection* e defina o *Driver Name* para "Interbase" e *Connection Name* para "Employee" (**Figura 1**). Confirme e prossiga para a configuração dos demais parâmetros, começando pela propriedade *Database* onde deverá ser informado o caminho da base de dados *Employee.fdb* (<caminho>\employee.fdb).

Figura 1. Criando uma nova conexão

Defina o usuário do Firebird para acessar a base de dados (SYSDBA) e a senha para o usuário (masterkey). Na propriedade *ServerCharSet* utilizaremos o padrão *Win1252* e em *SQLDialect* utilize o padrão "3" para que possamos utilizar comandos de *SQL* mais complexos (**Figura 2**).

Ainda no componente *sqlConexao*, defina sua propriedade *LoginPrompt* para *False* e altere a propriedade *Connected* para *True* testando assim o sucesso da conexão.

No primeiro exemplo prático iremos criar uma estrutura de acesso entre duas tabelas do banco de dados, *Departament* e *Employee*, onde as utilizaremos para criar nosso relacionamento *master/detail*.

Nessa estrutura iremos utilizar também alguns componentes da paleta *Data Access* para criar a estrutura de ligação. Por fim utilizaremos um componente *DBNavigator* e um *DBGrid ambos* da paleta *Data Controls* deixando toda a estrutura navegável e visível ao usuário.

Toda esta lógica irá também permitir que possamos inserir novos registros no *DataSet* pai, *sqldsDepartament*, e também no *DataSet* filho, *sqldsEmployee*. Realizaremos apenas um *ApplyUpdates* no *DataSet* pai, gravando os registros cadastrados nas duas tabelas do banco de dados.

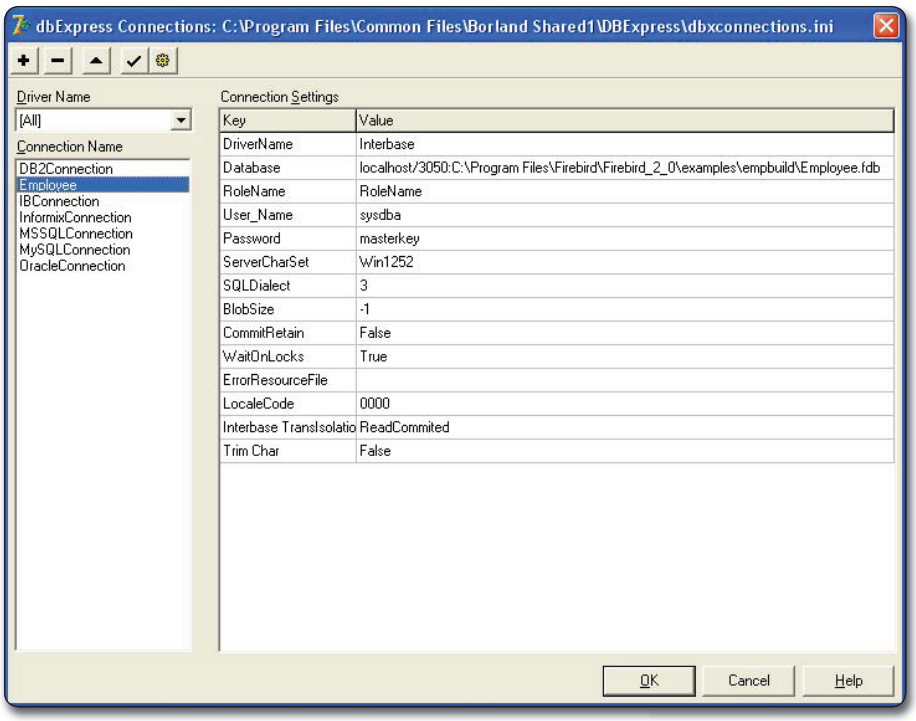


Figura 2. Configurando a conexão

Crie no seu *Data Module* uma estrutura conforme ilustrado na **Figura 3** alterando o nome dos componentes de acordo com a ilustração na imagem, observando na **Tabela 1** as configurações individuais e algumas dicas de uso para cada componente.

Convém lembrar que para um funcionamento correto de toda estrutura de componentes criada acima é necessário uma relação identificada entre a chave primária e estrangeira das tabelas, sendo obrigatória a igualdade de nomes entre o campo chave primária da tabela pai e a condição *Where* (chave estrangeira) da tabela filho.

Com um duplo clique sobre os componentes *cdsDepartament* e também *cdsEmployee* acesse no *Fields Editor* de ambos os itens listados e faça, de forma individual, a configuração da propriedade *ProviderFlags*, alterando os itens *pflnKey* e *pflnWhere* de acordo com o campo selecionado ser ou não um campo chave da tabela (*pflnKey* alterado para *True* ou *False*) e se o mesmo deverá ou não ser utilizado na condição *Where* em casos de *Update* de registros no componente (*pflnWhere* alterado para *True* ou *False*).

> **Nota:** Adicione ao uses do formulário principal a unit do *Data Module* através do menu *File>Use* unit ou *Alt* + F11.

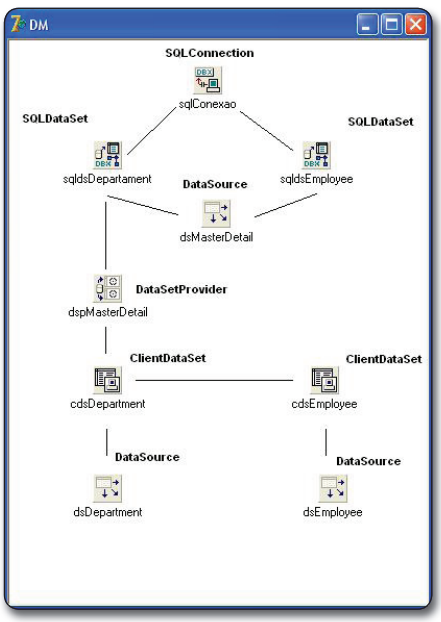


Figura 3. Estrutura de criação do master/detail

Componente	Configuração das propriedades
sqldsDepartament	SQLConnection: sqlConexao CommantText: select * from DEPARTAMENT
dsMasterDetail	DataSet: sqlConexao
sqldsEmployee	SQLConnection: sqlConexao CommantText: select * from EMPLOYEE where DEPT_NO = :DEPT_NO DataSource: dsMasterDetail Acesse na propriedade Params do componente o item "DEPT_NO" e altere sua propriedade DataType para "ftInteger". *Com um duplo clique abra o Fields Editor e adicione todos os campos obtidos da consulta SQL
dspDepartament	DataSet: sqldsDepartament
cdsDepartament	ProviderName: dspDepartament Active: True *Com um clique duplo abra o Fields Editor e adicione todos os campos da tabela. Note a presença de um novo campo chamado sqldsEmployee que fará o vínculo com a tabela filho.
dsDepartament	DataSet: cdsDepartament
cdsEmployee	DataSetField: cdsDepartamentsqldsEmployeee Active: True *Veja que neste caso não foi utilizada a propriedade ProviderName que foi substituída pela propriedade DataSetField e relacionada ao campo criado automaticamente em função do relacionamento máster/detail. Com um clique duplo abra o Fields Editor e adicione todos os campos da tabela.
dsEmployee	DataSet: cdsEmployee

Tabela 1. Configurando os componentes da estrutura master/detail

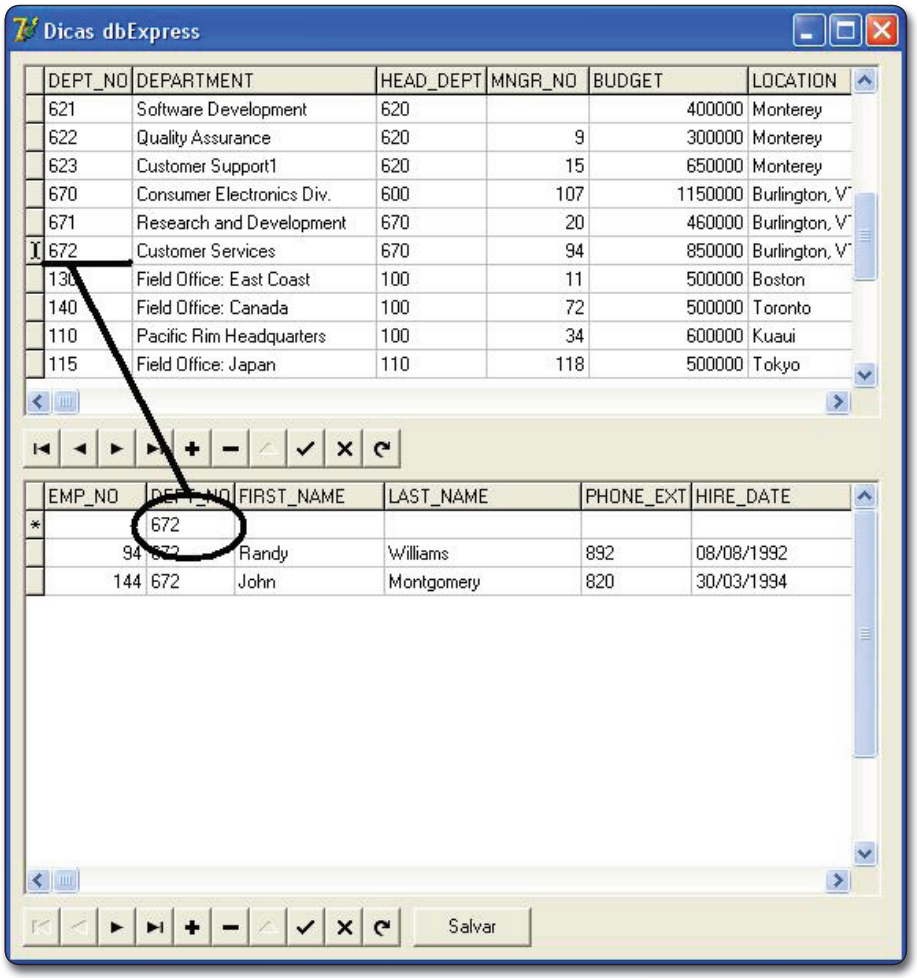


Figura 4. Inserção de registros nas tabelas

Para visualizar os resultados e registros de ambas as tabelas, volte ao formulário principal do sistema e adicione, da paleta *Data Controls*, dois componentes *DBGrid* e também dois componentes *DBNavigator*. Organize-os de modo que seja exibido um *DBGrid* seguido por um *DBNavigator*, repetindo a ordem para os demais. Selecione simultaneamente o *DBGrid1* e *DBNavigator1* e vincule sua propriedade *DataSource* ao *dsDepartament* do *Data Module* principal.

Selecione também os componentes *DBGrid2* e *DBNavigator2* e configure sua propriedade *DataSource* de modo que seja relacionado ao componente *dsEmployee* configurado no *DM*.

Adicione também um componente *BitBtn* alterando sua propriedade *Caption* para "Salvar Alterações" e atribua o seguinte código ao seu evento *OnClick:*

```
DM.cdsDepartament.ApplyUpdates(0);
```

Observe que a repercussão das informações é feita ao banco de dados somente no *ClientDataSet* pai (*cdsDepartament*), onde este se encarrega de salvar todas as demais informações que foram inseridas ou alteradas na estrutura de componentes que está a ele relacionada.

Essa é uma prática muito utilizada em aplicações com relacionamentos de 1:N (uma chave pai para vários registros filhos), principalmente em situações de compra ou venda, envolvendo assim os itens de produtos que são adicionados.

A utilização desta estrutura dispensa os cuidados tradicionais que o programador precisa prever em seu sistema ao inserir o registro de compra, salvá-lo na base de dados e liberando aí a estrutura de cadastro de itens, onde ainda deveria prever que o usuário pudesse optar por cancelar a operação.

Certifique-se de que ambos os componentes *ClientDataSet* estejam ativos e execute a aplicação. Faça a inserção de um novo registro no quadro de departamentos, preencha no *DBGrid* todos os campos solicitados, e sem salvar o registro pressione o botão *Add* no *DBNavigator2* relacionado à tabela *Employee*.

Observe que o campo *DEPT_NO* assume imediatamente o valor informado na tabela de departamentos, que ainda não foi salva, sendo assim, podemos observar

pelo *DBGrid* que a estrutura de departamentos permanece em modo de edição enquanto a estrutura de empregados está em modo de inserção de registros, que ao ser salvo irá aplicar ambos os registros dos quadros (**Figura 4**).

Podemos observar no exemplo criado que ao cancelar a operação em qualquer momento, todas as modificações em ambas tabelas são canceladas simultaneamente. Da mesma forma ao salvá-las, ambas tabelas são atualizadas automaticamente.

Estes mesmos processos e componentes podem ser usados também nas situações de exclusão e alteração de registros, lembrando apenas que no caso de uma exclusão é necessário excluir antes todos os relacionamentos do *master/detail* e após isso efetuar a exclusão do registro principal.

No caso de alteração basta reescrever as informações conforme desejado e salvar. Opte também por substituir o *DBGrid* por componentes *DBEdit*, entre outros e também criar sua estrutura customizada de botões para receber as instruções de manutenção do registros.

Listagem 1. Log salvo pelo componente SQLMonitor

```
INTERBASE - isc_attach_database
INTERBASE - isc_dsql_allocate_statement
INTERBASE - isc_start_transaction
select * from DEPARTMENT
INTERBASE - isc_dsql_prepare
INTERBASE - isc_dsql_describe_bind
INTERBASE - SQLDialect = 3
INTERBASE - isc_dsql_execute
INTERBASE - isc_dsql_fetch
INTERBASE - isc_dsql_allocate_statement
select * from EMPLOYEE where DEPT_NO =  ?
INTERBASE - isc_dsql_prepare
INTERBASE - isc_dsql_describe_bind
INTERBASE - SQLDialect = 3
INTERBASE - isc_dsql_execute
INTERBASE - isc_dsql_fetch
INTERBASE - isc_dsql_allocate_statement
SELECT 0, '', '', A.RDB$RELATION_NAME, A.RDB$INDEX_NAME, B.RDB$FIELD_NAME, B.RDB$FIELD_
POSITION, '', 0, A.RDB$INDEX_TYPE, '', A.RDB$UNIQUE_FLAG, C.RDB$CONSTRAINT_NAME,
C.RDB$CONSTRAINT_TYPE FROM RDB$INDICES A, RDB$INDEX_SEGMENTS B FULL OUTER JOIN
RDB$RELATION_CONSTRAINTS C ON A.RDB$RELATION_NAME = C.RDB$RELATION_NAME AND
C.RDB$CONSTRAINT_TYPE = 'PRIMARY KEY' WHERE (A.RDB$SYSTEM_FLAG <> 1 OR A.RDB$SYSTEM_
FLAG IS NULL) AND (A.RDB$INDEX_NAME = B.RDB$INDEX_NAME) AND (A.RDB$RELATION_NAME =
UPPER('DEPARTMENT')) ORDER BY RDB$INDEX_NAME
[…]
```

Listagem 2. Código do botão Salvar

```
procedure TfrmCountry.Button1Click(Sender: TObject);
begin
  if ((Length(Edit1.Text)>0) and
    (Length(Edit2.Text)>0)) then
  begin
    DM.cdsNovoCountry.Params[0].Value := Edit1.Text;
    DM.cdsNovoCountry.Params[1].Value := Edit2.Text;
    DM.cdsNovoCountry.Execute;
    DM.cdsVerCountry.Close;
    DM.cdsVerCountry.open;
  end;
end;
```

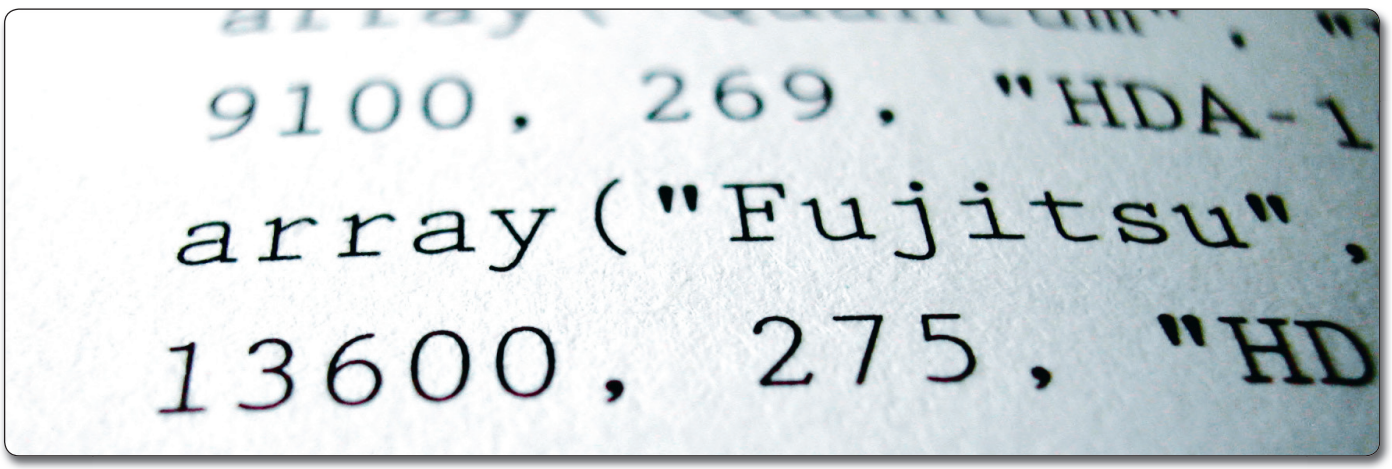
Monitorando os comandos SQL

Disponível na paleta *dbExpress* o componente *SQLMonitor* é geralmente utilizado durante o processo de desenvolvimento de um sistema. É através deste componente que o programador pode acompanhar todas as requisições e comandos que são trocados entre a aplicação e a base de dados. A funcionalidade do componente representa um *log* que será salvo durante todo o processo de execução do sistema gravando um passo a passo de tudo que está acontecendo.

De fácil configuração, adicione no *Data Module* um componente *SQLMonitor* da paleta *dbExpress*, ligue sua propriedade *SQLConnection* ao componente *sqlConexao*. Configure na propriedade *FileName* um diretório onde deseja que o *log* seja salvo, atribuindo a ele o nome "LogSQL.txt". Altere as propriedades *AutoSave* e *Active* do componente para *True*.

Compile e execute a aplicação. Observe que todas as instruções de *SQL* que configuramos nos componentes *SQLDataSet* foram enviadas ao banco de dados e assim capturadas pelo *log*, além de informações de dialeto de caracteres e comandos de busca na base. Faça algumas inclusões e/ou atualizações na aplicação e feche-a. Em seguida abra o *Windows Explorer*, localize o arquivo de *log* que criamos no diretório da aplicação e abra-o no *Bloco de Notas* ou no editor de textos que preferir.

Irá notar facilmente que toda a operação com o *BD* foi listada no arquivo de log.



Isso é importantíssimo quando necessitamos detectar ou mesmo corrigir um erro no sistema. (**Listagem 1**)

> **Nota:** Ao publicar sua aplicação em seus clientes desabilite as funcionalidades deste componente, pois evidentemente a ação do mesmo torna o rendimento da aplicação mais lenta.

Simplificando o uso de componentes

Em muitas situações acabamos nos perdendo na quantidade de componentes que estamos utilizando em nossas aplicações, tais como componentes de comunicação ao *BD*. São inúmeros os componentes *SQLDataSet*, *DataSetProvider* e *ClientDataSet*, alterando basicamente entre todos somente o comando de *SQL* que desejamos aplicar. Para este caso, uma forma de programação interessante seria simplificar o uso destes componentes, ou seja, imaginando uma estrutura onde você precisa mostrar no mesmo formulário os registros já cadastrados e deixar os campos para que um novo registro seja inserido, precisaríamos de apenas um *SQLDataSet*, um *DataSetProvider* e neste caso, dois componentes *ClientDataSet*, contrariando a estrutura convencional do *dbExpress*, ou seja, *SQLDataSet* e *DataSetProvider* individual para cada operação que deseja ser realizada.

Para ver na prática o funcionamento desta estrutura, adicione ao *Data Module* um componente *SQLDataSet*("sqldsGeral"), configure sua propriedade *SQLConnection* para o "sqlConexao" e deixe a propriedade *CommandText* em branco. Adicione também da paleta *Data Access* um componente *DataSetProvider*("dspGeral") relacionando sua propriedade *DataSet* ao "sqldsGeral". Modifique ainda na propriedade *Options* o item *poAllowCommandText* para *True*, que habilita a possibilidade de deixar todas as rotinas de *SQL* nos componentes *ClientDataSet* e estes por sua vez os transmite por meio do *DataSetProvider* ao *SQLDataSet* que se encarregará de executá-las na base de dados.

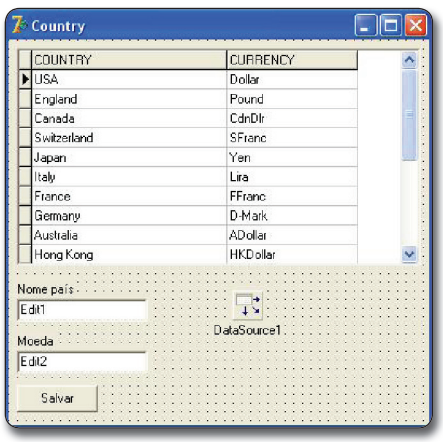


Figura 5. Formulário utilizando estrutura simplificada de componentes

Arraste para o *Data Module* dois componentes *ClientDataSet* da paleta *Data Access* nomeando-os como "cdsVerCountry" e "cdsNovoCountry" respectivamente. Ligue a propriedade *ProviderName* de ambos ao "dspGeral". Adicione à propriedade *CommandText* do *cdsVerCountry* a seguinte instrução *SQL* e ative-o:

```
select * from country
```

Na propriedade *CommandText* do *cdsNovoCountry* digite o comando *SQL* a seguir e logo após acesse a propriedade *Params* selecionando os parâmetros adicionados e altere sua propriedade *DataType* para *ftString:*

```
INSERT INTO COUNTRY (COUNTRY, CURRENCY)
VALUES (:COUNTRY, :CURRENCY)
```

Crie um novo formulário através do menu *File|New>Form*, altere sua propriedade *Caption* para "Country" e o *Name* para "frmCountry" e salve a *Unit* do formulário como "ufrmCountry.pas".

Acessando o menu *Project>Options* do Delphi altere o formulário principal da aplicação, selecionando o "frmCountry" e confirme. Adicione ao formulário a *Uses* do *Data Module* através do menu *File>Use unit* e logo após arraste para o formulário um componente *Data Source* da paleta *Data Access* ligando sua propriedade *DataSet* ao *cdsVerCountry* configurado no *Data Module*.

Adicione também um componente *DBGrid* organizando-o no topo do formulário e relacionando sua propriedade *DataSource* ao *DataSource1* recém configurado, onde os registros já poderão ser visualizados na tela.

Para a utilização do segundo *ClientDataSet* configurado e simulando um cadastro de países com as referentes moedas, adicione ao *frmCountry* dois *Labels* e dois *Edits*, alterando o *Caption* do *Label1* para "Nome País" e organize-o de modo que fique imediatamente acima do *Edit1*. Para o *Label2* altere a propriedade *Caption* para "Moeda" e organize-o logo acima ao *Edit2*. Por último adicione ainda um *Button* e altere o *Caption* para "Salvar" deixando o formulário semelhante ao ilustrado na **Figura 5** e logo após adicione ao evento *OnClick* do botão "Salvar" o código da **Listagem 2** que irá inserir um novo registro na tabela *COUNTRY* e exibí-lo no *Grid* acima.

Ao executar a aplicação, de imediato podemos observar todos os registros da tabela *COUNTRY* sendo listados no *DBGrid* resultantes do comando *SQL* que configuramos no *cdsVerCountry*. Através dos demais componentes, código *SQL* e parâmetros configurados no *cdsNovoCountry*, podemos inserir um novo registro, compartilhando nestes dois casos o mesmo *DataSetProvider* e o mesmo *SQLDataSet*.

> **Nota:** Ao distribuirmos aplicações que utilizam componentes da paleta dbExpress somos obrigados a distribuir juntamente com o executável as *DLL's Midas.dll* e *dbExpint.dll*. Uma dica importante é declarar as Units *DBExpint* e *MidasLib* que incorporam as mesmas ao executável.

Conclusão

Vimos neste artigo algumas dicas e facilidades durante o uso dos componentes da paleta *dbExpress* em conjunto com componentes de outras paletas em aplicações *Client/Server*.

Criamos uma forma de facilitar os controles de cadastros e consultas de registros em tabelas com relacionamentos de 1:N, além de monitoração de todos os comandos *SQL* que são enviados e recebidos do *BD*. Vimos ainda uma forma prática de reduzir e simplificar o uso de componentes na aplicação, dicas que serão muito úteis no prazo de construção da aplicação.

Procure estudar a arquitetura dos componentes *dbExpress*, como se comportam no dia-a-dia, performance e sua utilização com outros bancos de dados tais como: Oracle e MSSQL. ●

Nesta seção você encontra artigos intermediários sobre Delphi Win32 e Delphi .NET

Trabalhando com informações de páginas HTML

Aprenda uma maneira simples de captura de dados atualizados na Web

Luciano César Ascari
(luascari@gmail.com)
Mestrando em Engenharia de Software pela UFPR, Especialista em Desenvolvimento para Ambiente Internet: Orientação a Objetos, Java e Banco de Dados, em nível de Pós-Graduação Lato-Sensu pela UTFPR, graduação em Tecnologia em Informática - CEFET- PR. Atualmente é Gestor Técnico da Unidade Fisco-Contábil da Viasoft Informática. Tem experiência na área de Ciência da Computação, com ênfase em análise e desenvolvimento de software, em Delphi e Java.

A constante evolução dos sistemas de informação e as atualizações das informações existentes na internet fazem com que usuários busquem atualizar-se constantemente com dados disponíveis na grande rede mundial. Existem formas facilitadoras de obtermos informações da *Web*, uma delas é a busca de serviços de WebService. Através dos WebServices, serviços e informações são distribuídos de forma bem organizada. É bem comum que clientes de sistemas de informação, necessitem em seus sistemas *Desktop* de informações atualmente disponíveis somente em páginas Web, cujo acesso normal é via *browser*.

Com Delphi é perfeitamente possível consumir informações de qualquer WebService de maneira bastante simples. E não se trata de nenhuma inovação das últimas versões da ferramenta. A partir da versão 6.0 já é possível fazer uso dessa tecnologia. Como dito, em Delphi é possível mantermos conexão com um servidor Web e isso é feito via protocolo HTTP. Para esse propósito usaremos o componente *IdHttp* da paleta *IndyClients*. Na paleta indicada encontramos não somente o *IdHttp* como outros componentes para acesso a internet tais como: *IdFtp, IdSmtp, IdPop* etc.

Para exemplificar o uso da técnica em uma aplicação Delphi, criaremos um pequeno exemplo, onde faremos a visualização de resultados de concursos da *Mega-Sena*, *Dupla-Sena* e *Quina* da Caixa Econômica Federal.

Construindo a aplicação

O componente citado anteriormente possui algumas características interessantes como podemos ver a seguir:

- Propriedades
  - *AllowCookies :* permite a persistência por meio de *cookies* ou não, ou seja, informações recebidas da internet podem ser gravadas em forma de *cookies*;
  - *Port:* Número da porta que será utilizada para conexão;

• Métodos:

o *Get:* Retorna uma *string* utilizando o método *get* do protocolo HTTP contendo o conteúdo textual da página passada no para o parâmetro *Url*;

Ilustrando a utilização do componente *IdHttp* propõem-se a criação de um software que busca e exibição das informações dos últimos resultados da loteria a partir de uma página *Web*. Para o desenvolvimento desse software inicie uma aplicação criando um novo projeto em *File|New>Application*. Desenhe uma tela como no exemplo da **Figura 1**. Nomeie os componentes *Labels* da seguinte maneira:

• *Mega Sena:* "lbMegaConcurso", "lbMegaData" e "lbMegaResultado";

• *Dupla Sena:* "lbDuplaConcurso", "lbDuplaData", "lbDuplaResultado1" e "lbDuplaResultado2";

• *Quina:* "lbQuinaConcurso", "lbQuinaData" e "lbQuinaResultado";

Você pode personalizar os componentes da forma que melhor lhe agradar. Nesse exemplo apenas modifiquei para *Negrito* os componentes *Label* que receberão informações em tempo de execução. São ao todo três componentes *GroupBox*, dezenove *Labels*, um *Panel* e um *Button*. Além é claro do componente *IdHttp*("idHttp1") responsável pela busca das informações nas páginas *Web*.

> **Nota**: Salve a aplicação após desenhar a parte visual. Para este exemplo chamei o formulário de "frmPrincipal" e a *Unit* de "uPrincipal". Já o projeto chamei de "Loteria.dpr".

Inclua também um componente *Button*("bAtualizarResultados") com o *Caption* "Atualizar Resultados" no *panel* alinhado como *alBottom*.

Codificando o exemplo

Construída a tela, faremos a codificação do botão *bAtualizarResultados* que é responsável pela busca de informações e atualização da tela. Segue na **Listagem 1** a codificação correspondente ao evento *OnClick* do botão, sendo que a utilização do componente da classe *IdHttp* só foi necessária na chamada ao método *Get*.

Como pode ser visto na **Listagem 1**, declaramos uma variável do tipo *string* (*strHtml*) que por sua vez recebe o texto do arquivo *index2.php* do site *www.loteria.com.br*. O resultado na verdade é um arquivo HTML, ou seja, um texto contendo *tags*.

Note que por diversas vezes fizemos uso da função *GetValidTextFromTo*. Ela tem a finalidade de obter uma *sub-string* a partir do texto passado no primeiro parâmetro, delimitada por uma palavra representando a parte inicial do texto a considerar passada no segundo parâmetro. E uma *string* que delimita a parte final a considerar no terceiro parâmetro. Essa função foi desenvolvida especificamente para essa aplicação, e exemplificando, na chamada:

```
GetValidTextFromTo('Uma frase qualquer.',
  'Uma', 'qualquer.');
```

Temos como retorno o texto "frase". Declare a função *GetValidTextFromTo* como a seguir na cláusula *private* do formulário.

```
function GetValidTextFromTo(str, strIni,
strFin: string): string;
```

Em seguida pressione *Ctrl + Shift + C* para que o Delphi crie o cabeçalho da função na área *Implementation*. Digite o código da **Listagem 2** e observe como o esquema é bem simples.

Primeiramente guardamos nas variáveis *nIni* e *nFin*, ambas do tipo *Integer*, a posição de cada string (inicial e final) passados como parâmetro dentro do texto principal, nesse caso a própria página HTML. Caso *nFin* seja maior que zero, o que significa que conseguimos localizar

Figura 1. Formulário 1 com personalizações para o software

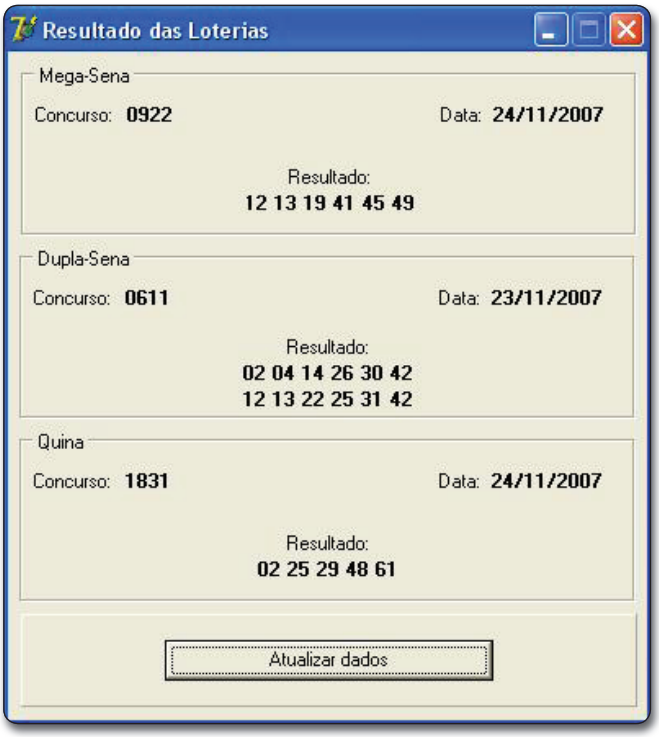


Figura 2. Aplicação rodando clique no botão "Atualizar Resultados"

no texto a *string* do primeiro parâmetro dentro do arquivo HTML, então envolvemos o texto encontrado em um laço *For*. Esse laço fará a cópia dos caracteres da string para o *Result* da função. Então retornamos os dados adquiridos e atualizamos os controles visuais.

Após toda a codificação basta a compilar a aplicação e executar o programa. A **Figura 2** exibe a aplicação rodando após clicarmos no botão "Atualizar Dados".

Conclusão

Concluindo, nesse artigo foi apresentada uma forma simples de busca de informações de uma página Web textual. O exemplo ilustrou uma aplicação simples que retorna informações atuais obtidas dos resultados de loteria expostos na Web.

Funcionalidade semelhante pode ser aplicada em outros contextos, atualizando os softwares *Desktop* com diversos dados atualizados das páginas Web, tais como cotação de moedas e valores de ações das bolsas. Até a próxima. ●

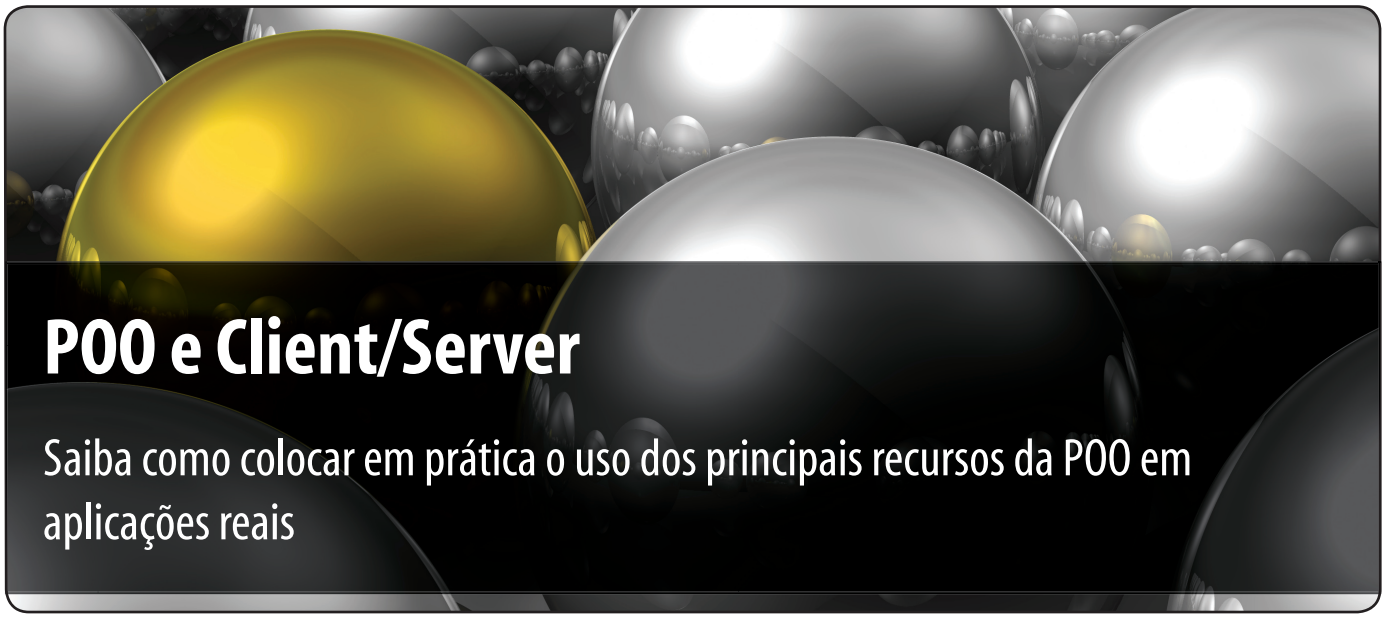
Listagem 1. Codificação do evento OnClick do botão "Atualizar Resultados".

```
procedure TfrmPrincipal.bAtualizarResultadosClick(Sender: TObject);
var
  strHtml, strSena : string;
  nPosIni, nPosFin : Integer;
begin
  { Carregando a página como string }
  strHtml := idHttp1.get('http://www.loteria.com.br/loteria/index2.php');
  { Carregando sub-string da Mega Sena, limpa de caracteres especiais}
  strSena := GetValidTextFromTo(strHtml,'<b>Mega Sena</b><br>', '<b>Dupla Sena</b><br>');
  { Setando valores aos campos da Mega Sena }
  lbMegaConcurso.Caption := GetValidTextFromTo(strSena,'', '-');
  lbMegaData.Caption := GetValidTextFromTo(strSena,'-', '<br>');
  lbMegaResultado.Caption := GetValidTextFromTo(strSena,
      '<span class="textoNumerosResultado">', '</span><br>');
  { Carregando sub-string da Dupla Sena, limpa de caracteres especiais }
  strSena := GetValidTextFromTo(strHtml, '<b>Dupla Sena</b><br>', '<b>Quina</b><br>');
  { Setando valores aos campos da Dupla Sena }
  lbDuplaConcurso.Caption := GetValidTextFromTo(strSena,'', '-');
  lbDuplaData.Caption := GetValidTextFromTo(strSena,'-', '<br>');
  lbDuplaResultado1.Caption := GetValidTextFromTo(strSena,
      '<span class="textoNumerosResultado">', '<BR>');
  lbDuplaResultado2.Caption := GetValidTextFromTo(strSena,'<BR>','</span><br>');
  { Carregando sub-string da Quina, limpa de caracteres especiais }
  strSena := GetValidTextFromTo(strHtml, '<b>Quina</b><br>', '<b>LotoMania</b><br>');
  { Setando valores aos campos da Quina }
  lbQuinaConcurso.Caption := GetValidTextFromTo(strSena,'', '-');
  lbQuinaData.Caption := GetValidTextFromTo(strSena,'-', '<br>');
  lbQuinaResultado.Caption := GetValidTextFromTo(strSena,
      '<span class="textoNumerosResultado">', '</span><br>');
end;
```

Listagem 2. Implementação da função GetValidTextFromTo

```
function TfrmPrincipal.GetValidTextFromTo(str, strIni, strFin: string): string;
var
  i, nIni, nFin : Integer;
  cChar : PChar;
begin
  Result := '';
  nIni := Pos(strIni, str);
  nFin := Pos(strFin, str);
  if (nFin > 0) then
  begin
    { Apontando para o endereço do primeiro caracter da sub-string}
    cChar := Addr(str[nIni + Length(strIni)]);
    for i := nIni + Length(strIni) to nFin-1 do
    begin
      { Verificando se o char do endereço não trata-se de um caracter indesejável }
      if (not (cChar^ in  [#0, #13, #9, #10])) then
        Result := Result + cChar^;
      Inc(cChar);
    end;
    { Retira espaços indesejáveis }
    Result := Trim(Result);
  end;
end;
```

Nesta seção você encontra artigos intermediários sobre Delphi Win32 e Delphi .NET



POO e Client/Server

Saiba como colocar em prática o uso dos principais recursos da POO em aplicações reais

Maikel Marcelo Scheid
(maikelscheid@gmail.com)
é técnico em Informática com ênfase em Análise e Programação de Sistemas. Atua na área de Desenvolvimento de Softwares em Delphi para plataforma Win32 e .NET com banco de dados Firebird e MS SQL há 3 anos.

Veremos neste artigo uma aplicação simplificada do uso da Programação Orientada a Objetos (POO) em classes de negócios através de uma aplicação *Client/Server*. Estudaremos algumas estruturas de programação utilizadas e, dentre elas, uma das principais características da POO: a capacidade de reutilização de códigos.

Criaremos uma *Unit* de funções com códigos genéricos, que não serão utilizados em apenas uma aplicação, mas que se empregarão perfeitamente na grande maioria dos nossos sistemas *Client/Server*, seja com funções de manipulação de formulários, atualização de registros em componentes de acessos a dados ou qualquer outra funcionalidade.

Para usar os recursos da POO precisamos entender e planejar muito bem nossas aplicações, estudar bem as regras de negócios e assim construir estruturas de códigos que poderão simplificar em muito nosso trabalho em casos de alterações de estrutura e manutenções do sistema.

Tomaremos como exemplo no decorrer deste artigo uma pequena aplicação desenvolvida em BDS 2006 com o acesso a uma estrutura de dados do Firebird 1.5 simulando um cadastro de produtos.

Utilizaremos a classe de uma forma simples em um exemplo prático onde criaremos métodos encapsulados para realizar cálculos referentes aos valores da simulação de itens de uma venda e também do fechamento da venda em relação à devolução de troco se assim necessário.

Conhecendo a estrutura de uma classe (*class*)

A estrutura de uma classe é definida por um conjunto de membros que definem o estado e o comportamento de um objeto geralmente implementando métodos e atributos. Os métodos de uma classe são geralmente divididos em públicos ("Public") e privados ("Private"). Os métodos públicos são de livre acesso durante seu

uso. Já os privados são acessados somente internamente pela classe, ou seja, ficam encapsulados.

Nas definições desse conjunto de peculiaridades que formam uma classe, podemos observar a seguinte estrutura:

Unit : informa que o arquivo é referente a uma unidade seguida do nome da mesma.

Interface: estabelece a fronteira de comunicação entre a classe e o ambiente externo. É aqui também que declaramos nossos novos tipos e métodos ("Procedimentos ou Funções").

Uses: esta cláusula permite listar e utilizar objetos da nossa pasta de trabalho. Quando declaramos essa cláusula numa determinada parte da unidade estamos dizendo ao Delphi que poderemos utilizar os métodos e/ou propriedades que o objeto da lista proporciona acesso.

Type: com esta cláusula podemos declarar novos tipos e/ou classes.

Implementation: é a partir desta cláusula que codificamos nossos métodos, funções e procedimentos. Nesta seção também podemos declarar novos tipos e constantes.

Dentro desta mesma estrutura ainda poderemos ter a herança de classes, onde uma classe pode ser criada a partir de outra já existente e herdar os atributos e comportamentos da estrutura.

Criando o banco de dados

Para iniciarmos nosso exemplo prático, criaremos uma estrutura simplificada de apenas uma tabela, *Produtos* (**Figura 1**).

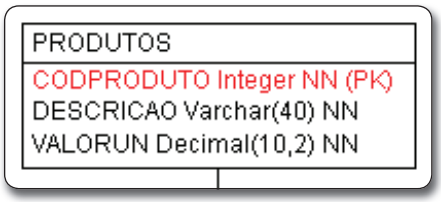


Figura 1. Diagrama ER da tabela

Listagem 1. Script de criação da base de dados

```
SET SQL DIALECT 3;
SET NAMES Win1252;

CREATE DATABASE 'C:\SISVendas\dbvendas.fdb'
USER 'SYSDBA' PASSWORD 'masterkey'
PAGE_SIZE 4096
DEFAULT CHARACTER SET Win1252;

CREATE TABLE PRODUTOS  (
 CODPRODUTO Integer NOT NULL,
 DESCRICAO Varchar(40) NOT NULL,
 VALORUN Decimal(10,2) NOT NULL,
 Primary Key (CODPRODUTO)
);
```

A partir desta estrutura de dados iremos utilizar os códigos orientados a objetos.

Definida a estrutura no programa de modelagem de dados, realize a exportação do script da **Listagem 1** e utilize o gerenciador de banco de dados para então fazer a importação e criação da base de dados. Crie sua base com o nome de *"dbvendas.fdb"*.

Com o banco de dados já criado, acesse a tabela de "Produtos" e crie um autoincremento na chave primária "CODPRODUTO". Depois de realizada esta configuração, nossa base de dados estará apta a ser utilizada e passaremos agora à criação da nossa aplicação de cadastros e consultas. Utilizarei neste artigo a BDS 2006, mas fique à vontade para escolher outra versão.

Criando a aplicação

Crie uma nova *VCL Forms Application – Delphi for Win32* no menu *File* do BDS 2006 e salve o projeto no caminho "C:\SISVendas\" com o nome de *"prjVendas .bdsproj"*. Salve a *Unit* principal como *"uPrincipal. pas"*. Altere as seguintes propriedades do formulário principal:

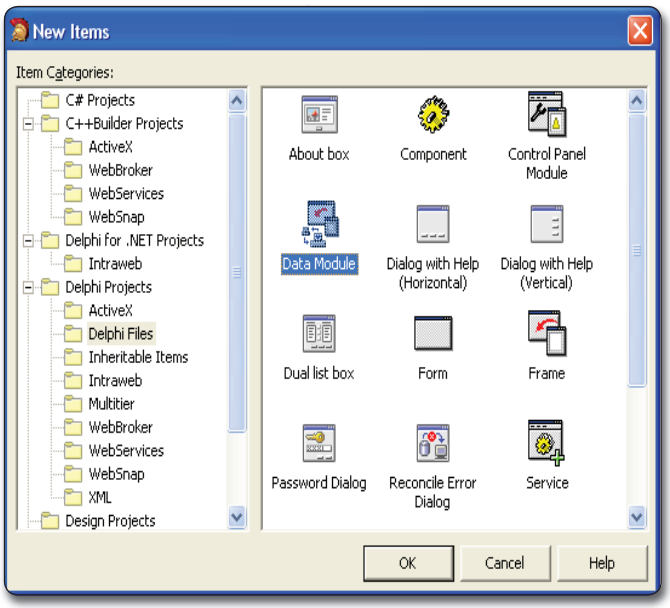


Figura 2. Adicionando o DataModule ao projeto

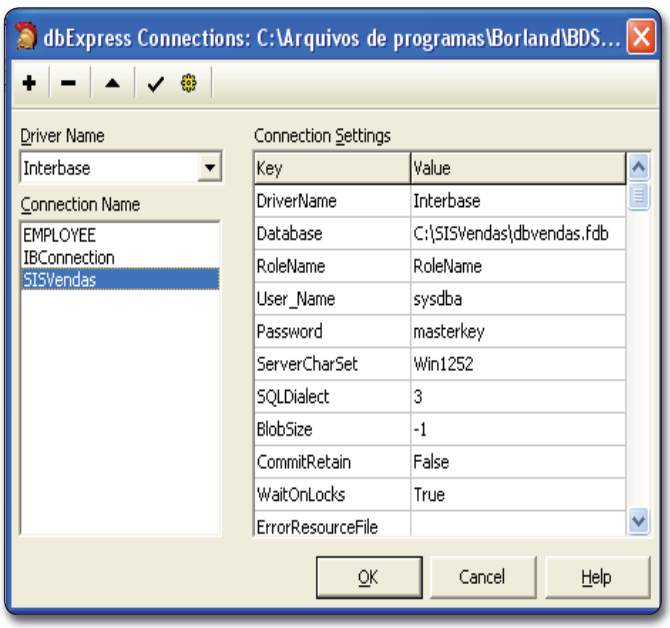
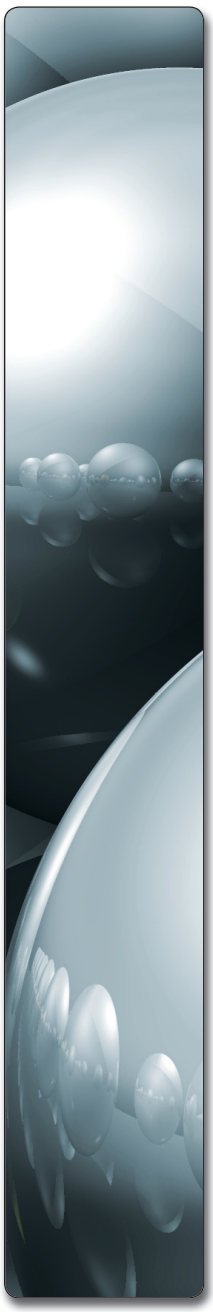


Figura 3. Configurando o acesso à fonte de dados



No formulário principal altere o seu *Caption* para "Sistema de Vendas" e marque a opção *biMaximize* como *False* na propriedade *BorderIcons*. Em seguida altere a propriedade *Name* para "frmPrincipal" e o *KeyPreview* para *True*.

Adicione agora a sua aplicação um *Data Module* o qual utilizaremos para concentrar a conexão com o banco de dados e também os componentes de consultas e manipulação dos registros. Acesse o menu *File|New>Other>Delphi Files* e selecione *Data Module* e confirme (**Figura 2**).

Com o *Data Module* já criado, altere seu nome para "DM" e salve a *Unit* como "uDM.pas". Arraste para o mesmo um componente *SQLConnection* da paleta *dbExpress*, altere o nome para "sqlCONEXAO". Com um duplo clique sobre o mesmo faça a configuração de maneira que seja possível conectar-se ao banco "dbvendas.fdb" criado anteriormente como na **Figura 3**.

Realizada a configuração anterior faremos a configuração dos componentes de cadastros de produtos. Adicione um componente *SQLDataSet* da paleta *dbExpress ("sdsProdutos")* apontando-o para o *sqlConexao* usando a propriedade *SQLConnectoin*. Em seguida digite a seguinte instrução *SQL* na propriedade *CommandText:*

```
select CODPRODUTO, DESCRICAO, VALORUN
from PRODUTOS where CODPRODUTO = :CODPRODUTO
```

Note que estamos usando uma instrução *SQL* simplescom a passagem de um parâmetro, "CODPRODUTO". Essa estrutura de programação em aplicações *Client/Server* é muito importante, pois evita de trazermos muitos registros em um resultado quando queremos apenas um registro. Qual seria a necessidade de carregar esses 20 mil registros de uma única vez na memória de um *ClientDataSet* se precisaremos de apenas 1 registro por vez? Nenhuma. Criaremos um formulário de pesquisa onde o usuário irá buscar o dado que deseja alterar, e tão somente este será carregado na memória do *ClientDataSet*. São estas as técnicas de programação que dão performance a sua aplicação.

Precisamos agora configurar o parâmetro adicionado, por isso acesse a propriedade *Params* do *sdsProdutos* e selecione o parâmetro CODPRODUTO alterando sua propriedade *DataType* para *ftInteger*. Com um duplo clique sobre o componente abra o *Fields Editor* e adicione todos os campos da consulta.

Selecione o campo CODPRODUTO e altere sua propriedade *ProviderFlags* marcando como *True* a opção *pfInkey*. Aproveite e modifique também a propriedade *Required* para *False*.

Agora configure o campo DESCRICAO alterando sua opção *pfInWhere* em *ProviderFlags* para *False*. Repita a última configuração para o campo VALORUN e por último marque a propriedade *Currency* para *True*.

Feche o *Fields Editor* e adicione agora ao DM um componente *DataSetProvider* ("dspProdutos"). Relacione sua propriedade *DataSet* ao *sdsProdutos*. Adicione também um *ClientDataSet* ("cdsProdutos") e relacione a propriedade *ProviderName* ao *dspProdutos*.

Uma configuração importante é trazer os parâmetros do *SQLDataSet* para que possam ser configurados diretamente no *ClientDataSet*, para isso clique com o botão direito do mouse sobre o *ClientDataSet* e selecione a opção *Fetch Params*. Observe na propriedade *Params* que agora o parâmetro CODPRODUTO está configurado no *ClientDataSet*. Abra o *Fields Editor* e adicione todos os campos da seleção assim como foi feito no *sdsProdutos*.

Crie agora um novo formulário usando o menu *File|New>Form*. Salve-o como "uProdutos.pas" e altere o nome do *formulário* para "frmProdutos". Adicione um *Button* ao formulário e altere sua propriedade *Caption* para "Pesquisar". Volte ao DM, abra o *Fields Editor* do *cdsProdutos* selecionando todos os campos e arrastando-os para o *frmProdutos*. Inclua um componente *DBNavigator* conectando sua propriedade *DataSource* ao *DataSource1* automaticamente adicionado pelo Delphi. Selecione o *DBEdit1* e marque *True* na sua propriedade *ReadOnly*. Arranje os componentes no formulário conforme **Figura 4.**

> **Nota:** Não esqueça de criar o auto-incremento na tabela de PRODUTOS no banco de dados.

No evento *OnShow* do formulário digite o código a seguir, que será responsável por abrir a conexão com o banco quando o formulário for chamado:

```
DataSource1.DataSet.Open;
```

Da mesma forma é necessário fechar o *DataSet* no evento *OnClose* do formulário, para isso digite o código a seguir no evento citado:

```
DataSource1.DataSet.Close;
```

Usando Units

Acabamos de criar nosso *formulário*, mas que ainda não está sendo chamado

Listagem 2. Implementação da procedure de criação de formulários

```
procedure CriarFormulario(FormClasse: TComponentClass;
  NomeForm : TForm);
begin
  Application.CreateForm(FormClasse,NomeForm);
  try
    NomeForm.ShowModal;
  finally
    NomeForm.Free;
  end
```

Figura 4. Formulário de Produtos

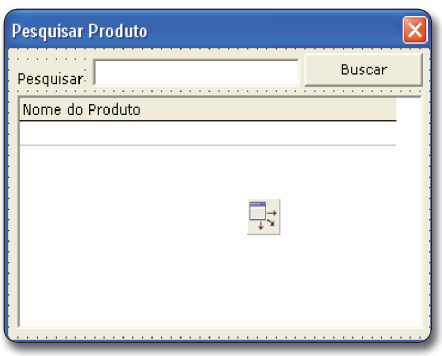


Figura 5. Formulário de pesquisa de produtos

em nenhuma outra *Unit*. Para realizar esta chamada iremos criar uma *Procedure* que irá criar o formulário e dar o *ShowModal* ao mesmo. Você poderá usar esta *Procedure* não somente com a janela de produtos, mas sim com todos os demais formulários do seu sistema.

Para começar adicione uma nova *Unit* ao projeto usando a opção *File|New>Other>Delphi Files>Unit*. Salve-a como "uFuncoes.pas" e declare as *Units Forms* e *Classes a Uses da Unit criada* logo abaixo a declaração *Interface*. Em seguida adicione a seguinte *Procedure* logo abaixo de *Uses:*

```
procedure CriarFormulario(FormClasse:
TComponentClass; NomeForm : TForm);
```

Na seção *Implementation* realize a codificação da *Procedure*, adicionando o código da **Listagem 2.**

Volte agora até o formulário principal do sistema, adicione ao topo um *Button*, altere sua propriedade *Caption* para "Produtos" e adicione a seguinte linha de código no evento *OnClick* do *Button:*

```
CriarFormulario(TfrmProdutos,frmProdutos);
```

Perceba que nesta linha de código estamos passando a classe e o nome do formulário que queremos criar. Ao executar a aplicação e clicar sobre o botão de produtos veja que o *formulário* é criado em *runtime*.

> **Nota:** Acesse o menu *Project>Options* e retire o formulário de produtos da lista de criação automática do projeto e mova o *Data Module* para cima de maneira que seja o primeiro a ser carregado pelo sistema.

Provavelmente você deve ter tentado realizar o cadastro de um produto, que aparentemente tenha funcionado, mas o registro não consta no banco de dados, pois o componente *DBNavigator* apenas realiza a ação de *Post*, mas precisamos aplicar estes registros na base de dados. Para isso volte a *Unit* de funções e crie uma nova *Procedure* que receba como parâmetro um objeto do tipo *ClientDataSet* e codifique-a como a seguir. Declare *DBClient* ao *Uses* da *Unit* de funções.

```
procedure AplicarRegistro(NomeCDS :
TClientDataSet);
begin
  NomeCDS.ApplyUpdates(-1);
end;
```

A *Procedure* anterior recebe como parâmetro o nome de um *ClientDataSet*, que recebe um *ApplyUpdates* fazendo a gravação dos registros no banco de dados. Volte ao *DM* e adicione no evento *AfterPost* e *AfterDelete* do *cdsProdutos* a seguinte linha de código:

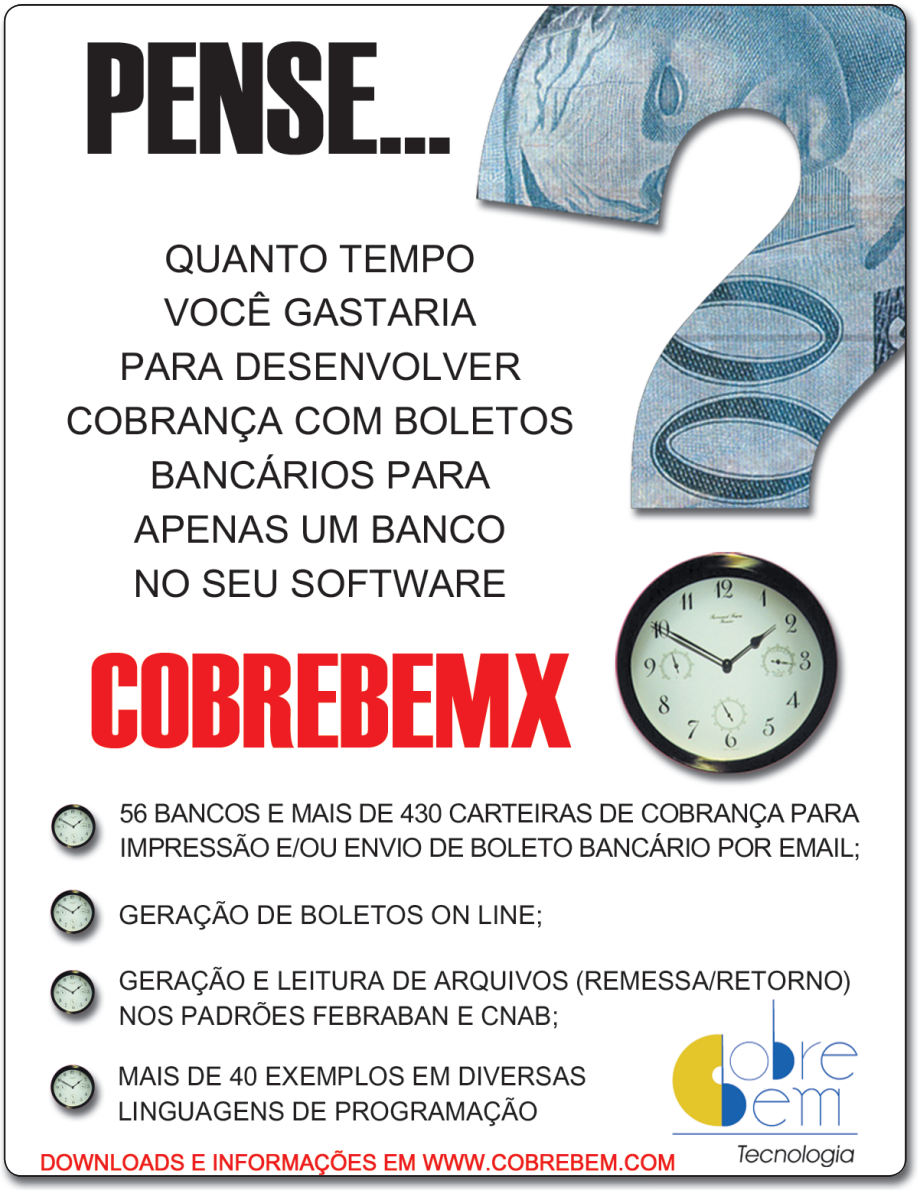
```
AplicarRegistro(TClientDataSet(DataSet));
```

Nesse momento ao cadastrar um produto verá que o mesmo foi corretamente adicionado ao banco de dados. Faremos agora um novo formulário o qual usaremos para pesquisar os produtos para a alteração. Adicione um novo *formulário*, salve-o como "uSearchProduto.pas" e altere sua propriedade *Caption* para "Pesquisa de Produtos".

Vá até o *Data Module* e inclua um novo trio de componentes sendo: um novo *SQLDataSet* ("sdsPesqProduto") com a propriedade *SQLConnection* igual a *sqlConexao* e *CommandText* com a instrução *SQL* a seguir:

```
select CODPRODUTO, DESCRICAO from PRODUTOS
where Upper(descricao) like :descricao
```

Configure na propriedade *Params* o parâmetro DESCRICAO como *DataType = ftString*. Insira um *DataSetProvider* ("dspPesqProduto") modificando sua propriedade *DataSet* para *sdsPesqProduto*. E por último inclua um novo *ClientDataSet* ("cdsPesqProduto") modificando sua propriedade *ProviderName* para *dspPesqProduto*. No *ClientDataSet* clique com



Listagem 3. Implementação dos códigos da criação da classe

```
unit uClassPVendas;
interface
uses
  SysUtils;
type
  TVendas = class
  { Cria a classe para calcular os ítens da Venda }
  private
    FQuantidade : Real;
    FValorItem  : Real;
    FTotalItem  : Real;
    procedure SetQuantidade(const Value: Real);
    procedure SetTotalItem(const Value: Real);
    procedure SetValorItem(const Value: Real);
  public
    procedure SomaValorTotalItem;
    property Quantidade : Real read fQuantidade write setQuantidade;
    property ValorItem : Real read fValorItem write setValorItem;
    property TotalItem : Real read fTotalItem write setTotalItem;
  end;
TValores = class
  { Cria a classe para calcular devolução de troco }
  private
    FTotalVenda : Real;
    FDinheiroRecebido : Real;
    FTroco : Real;
    FDesconto : Real;
    procedure SetDesconto(const Value: Real);
    procedure SetDinheiroRecebido(const Value: Real);
    procedure SetTotalVenda(const Value: Real);
    procedure SetTtroco(const Value: Real);
  public
    procedure CalcularTroco;
    property TotalVenda : Real read fTotalVenda write setTotalVenda;
    property DinheiroRecebido : Real read fDinheiroRecebido write setDinheiroRecebido;
    property Troco : Real read fTroco write setTroco;
    property Desconto : Real read fDesconto write setDesconto;
  end;
implementation
{ TVendas }
procedure TVendas.SetQuantidade(const Value: Real);
begin
  if Value > 0 then
    fQuantidade := Value
  else
  raise
  Exception.Create('Quantidade deve ser maior que 0');
end;
procedure TVendas.SetTotalItem(const Value: Real);
begin
  FTotalItem := Value;
end;
procedure TVendas.SetValorItem(const Value: Real);
begin
  if Value > 0 then
    fValorItem := Value
  else
  raise
  Exception.Create('Valor deve ser maior que R$0,00');
end;
procedure TVendas.SomaValorTotalItem;
begin
  try
    FTotalItem := FValorItem * fQuantidade;
  except
    on E:Exception do
    raise
    Exception.Create('Erro durante o cálculo');
  end;
end;
{ TValores }
procedure TValores.CalcularTroco;
begin
  Troco := DinheiroRecebido - (TotalVenda - Desconto);
end;
procedure TValores.setDesconto(const Value: Real);
begin
  FDesconto := Value;
end;
procedure TValores.setDinheiroRecebido(const Value:
  Real);
begin
  FDinheiroRecebido := Value;
end;
procedure TValores.setTotalVenda(const Value: Real);
begin
  FTotalVenda := Value;
end;
procedure TValores.setTroco(const Value: Real);
begin
  FTroco := Value;
end;
end.
```

o botão direito do mouse e selecione a propriedade *Fetch Params*.

Volte agora ao formulário, adicione um *Label* com o *Caption* "Pesquisar" e ao lado coloque um *Edit* com a propriedade *Text* em branco. Adicione um *Button* com o *Caption* "Buscar". Insira um componente *DBGrid* da paleta *Data Controls* e em seguida um *DataSource*.

Pressione *ALT + F11* e inclua a *Unit* do *Data Module* ao *Uses* do formulário de pesquisa desta forma poderá ligar o *DataSource* inserido diretamente ao *cdsPesqProdutos* pela propriedade *DataSet*.

Relacione também a propriedade *DataSource* do *DBGrid* ao *DataSource1* recém adicionado. Organize os componentes no formulário conforme **Figura 5**.

No evento *OnClick* do *Button* faça a seguinte codificação, que será responsável por passar o valor digitado ao parâmetro do componente anteriormente configurado e realizar a pesquisa:

```
DM.cdsPesqProdutos.Close;
DM.cdsPesqProdutos.Params[0].AsString :=
  UpperCase(Edit1.Text)+'%';
DM.cdsPesqProdutos.Open;
```

Agora no evento *OnDblClick* do *DBGrid* codifique conforme a seguir. O código será responsável por passar o código do produto selecionado ao parâmetro do primeiro *ClientDataSet* que configuramos habilitado para trazer somente um registro por vez:

```
DM.cdsProdutos.Close;
DM.cdsProdutos.Params[0].AsInteger :=
  DM.cdsPesqProdutosCODPRODUTO.AsInteger;
DM.cdsProdutos.Open;
Close;
```

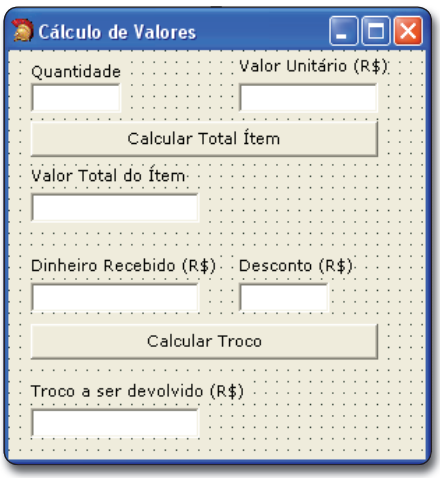


Figura 6. Formulário para simulaçao de venda

Antes de testar novamente a aplicação faça a chamada ao formulário de pesquisa usando o código a seguir no evento *OnClick* do botão de pesquisa.

```
CriarFormulario(
  TfrmSearchProduto,frmSearchProduto);
```

Veja que novamente estamos utilizando a mesma função que utilizamos para chamar antes o formulário de produtos, porém agora passando como parâmetro as definições para criação do formulário de pesquisa de produtos.

Da mesma forma que criamos essas *Procedures* você poderá implantar vários outros métodos orientados a objeto durante sua programação e assim reutilizar os códigos nesta aplicação ou ainda reutilizar toda a Unit de funções para demais projetos.

Usando classes de negócios

Para mostrar os benefícios da utilização de classes de negócios em sua programação, simularemos uma venda com itens, onde iremos nos beneficiar de regras concentradas na nossa classe para o cálculo dos valores dos itens e também valores do fechamento da compra.

No seu projeto adicione uma nova *Unit* e salve-a como "uClassPVendas.pas". Adicione a *Unit* o código da **Listagem 3**, onde na mesma estrutura estamos declarando duas classes ("TVendas" e "TValores") e cada uma delas com suas devidas variáveis e métodos encapsulados com somente as propriedades visíveis ao ambiente exterior da classe.

Note que em algumas *procedures* estamos verificando os valores recebidos e se não atenderem a regra estabelecida, uma exceção será criada. Você poderá fazer inúmeros testes com os valores recebidos, o que irá dar performance às suas regras de negócios.

Para implantar a utilização da classe no projeto, criaremos um formulário onde iremos digitar os valores, mas que nas suas aplicações certamente serão valores resultantes da consulta de itens de uma base de dados.

Criado um novo formulário, salve-o como "uCalcValores.pas", altere seu nome para "frmCalcValores" e o *Caption* para "Calculo de Valores". Coloque seis componentes *Label* ao formulário, seis componentes *Edit* e dois *Buttons*. Altere o nome dos componentes de acordo com a nomenclatura de cada *Label* e organize-os conforme **Figura 6**.

Listagem 4. Implementação dos códigos para cálculo de valores do item

```
procedure TfrmCalcValores.Button1Click(Sender: TObject);
var
  Item : TVendas;
begin
  Item := TVendas.Create;
  Item.Quantidade := StrToFloat(edtQuantidade.Text);
  Item.ValorItem  := StrToFloat(edtValorUnitario.Text);
  Item.SomaValorTotalItem;
  edtTotalItem.Text := FormatFloat('#0.00',Item.TotalItem);
end;
```

Listagem 5. Implementação dos códigos para cálculo de devolução de troco

```
procedure TfrmCalcValores.Button2Click(Sender: TObject);
var
  Valor : TValores;
begin
  Valor := TValores.Create;
  Valor.TotalVenda := StrToFloat(edtTotalItem.Text);
  Valor.DinheiroRecebido := StrToFloat(edtDinheiroRecebido.Text);
  Valor.Desconto := StrToFloat(edtDesconto.Text);
  Valor.CalcularTroco;
  edtTroco.Text := FormatFloat('#0.00',Valor.Troco);
end;
```

> **Nota:** Pressione *ALT + F11* para incluir a *Unit* de funções de cálculo ao *Uses* do formulário de cálculo.

Vamos iniciar a codificação pelo botão *Calcular Total do Item incluindo em seu evento OnClick* o código da **Listagem 4**.

Observe que estamos criando uma variável com o tipo da classe ("TVendas"), e na implementação da *Procedure* estamos dando um *Create* na mesma. Após criada conseguimos usar todas as propriedades e métodos que definimos como públicos na classe.

Você poderá notar que nenhum dos métodos que estão encapsulados são listados ao usarmos a classe. No código anterior, estamos passando os valores para as propriedades da classe, convertemos o texto digitado nos *Edits* para *Float* e passamos para cada propriedade.

Antes de jogar o resultado de volta ao *Edit*, executamos o método responsável pelo cálculo criado na classe. Ao testar sua aplicação, faça um teste digitando uma quantidade de produtos e informe o valor zero e veja a forma com que a exceção será tratada.

Da mesma forma faremos uso da outra classe criada na estrutura, atribuindo o código da **Listagem 5** ao evento *OnClick* do botão de cálculo do troco, onde também criaremos uma variável instanciada ao objeto da classe, dando um *Create* na implementação do método e então passando a usar suas propriedades passando-lhes os devidos valores.

Em seguida chamamos o método para realizar o cálculo, retornando assim através de outra propriedade da classe a quantidade de troco que deverá ser devolvida.

Você poderá ainda fazer outras verificações na classe, como exemplo se a quantidade de dinheiro recebida for suficiente para pagar o valor total da venda, se o valor do desconto estiver em determinada escala de porcentagem, entre outras várias funções que irão definir as regras do negócio para a venda de itens.

Perceba que essa classe poderá ser utilizada de várias formas diferentes, e que o principal benefício é que os códigos ficam centralizados em um único espaço, diminuindo o tempo de manutenção do sistema.

Conclusão

Vimos nesse artigo de forma simples exemplos práticos que irão trazer mais credibilidade e rapidez no processo de programação. Vimos uma forma de como reutilizar códigos de métodos orientados a objetos, o que irá evitar o seu re-trabalho em fazer um sistema com funções internas semelhantes, faz-se uma única função e utiliza-se em todo o resto da aplicação.

No uso de classes, vimos uma forma simples de centralizar as regras de negócio, onde em casos de manutenção de software você terá bons ganhos em relação ao uso dessa técnica de desenvolvimento. ●

Nesta seção você encontra artigos intermediários sobre Delphi Win32 e Delphi .NET

Dicas de RAVE Reports

Veja como criar relatórios em RAVE reports, configurá-los e exportá-los para PDF

Maikel Marcelo Scheid
(maikelscheid@gmail.com)
é técnico em Informática com ênfase em Análise e Programação de Sistemas. Atua na área de Desenvolvimento de Softwares em Delphi para plataforma Win32 e .NET com banco de dados Firebird e MSSQL. É membro da Equipe Editorial ClubeDelphi.

Utilizado para exibir e reportar resultados de buscas, cálculos, gráficos, entre inúmeras outras funcionalidades, os relatórios são indispensáveis aos nossos sistemas. Geralmente para cada tela de cadastro, temos no mínimo um relatório para exibir os registros, e que podem ser exibidos de forma contínua e total, ou em relatórios customizados com informações resultantes a uma busca na base de dados.

Partindo desta metodologia, veremos no decorrer deste artigo a utilização de um dos geradores de relatórios mais utilizados: RAVE Reports. Para que possamos perceber de forma concreta algumas das dicas, utilizaremos o Delphi 7 para criar uma aplicação com acesso a banco de dados e após isso veremos formas de customização e apresentação do *layout* de registros em *runtime*, linhas "zebradas"e também como criar parâmetros para destacar registros no corpo do relatório.

Acessando a base de dados *Employee.fdb*, que acompanha a instalação do *Firebird*, criaremos também um relatório que será salvo diretamente em arquivo via programação. Iremos salvar o mesmo em uma extensão .pdf, que poderá ser enviado diretamente neste formato ao seu cliente.

Criando a aplicação

Neste artigo utilizarei o Delphi 7 para criação da aplicação e demonstração dos exemplos e dicas de relatórios, mas fique à vontade para utilizar outra versão do Delphi para criação do projeto na plataforma Win32. Crie uma nova aplicação através do menu *File|New>Application*. Altere a propriedade *Name* do formulário inicial para "frmPrincipal" e a propriedade *Caption* para "Dicas avançadas de RAVE Reports". Salve a *Unit* principal do projeto como "uPrincipal.pas" e o projeto salve como "prjRelatorios.dpr".

Na primeira parte deste artigo veremos a criação de um relatório no RAVE Re-

ports, onde através de uma pesquisa *SQL* ao banco de dados *Employee.fdb* que será configurado a seguir, selecionaremos os registros referentes à tabela *Employee*. Serão então exibidos e configurados no relatório, de forma que a primeira formatação será a alternação entre as cores de cada bloco de registros (zebrar linhas). Faremos um tratamento através de uma condição *if* criada em "*Rave Language*" onde definiremos um parâmetro de valor do salário do funcionário. Esse parâmetro fará com que todos os valores excedentes tenham sua cor alterada.

Para criar a conexão ao banco de dados e estruturar os componentes de pesquisas, utilizaremos um *Data Module*, que deverá ser criado através do menu *File|New>Data Module*. Altere seu nome para "DM" e salve a *Unit* como "uDM.pas". Adicione um componente *SQLConnection* da paleta *dbExpress* e altere seu nome para "sqlConexao".

Para configurar os parâmetros de acesso ao banco de dados, dê um duplo clique sobre o componente *sqlConexao* e crie uma nova conexão utilizando o ícone de *Add Connection*, definindo o *Driver Name* para "Interbase" e *Connection Name* para "Employee" (**Figura 1**). Confirme e prossiga para a configuração dos demais parâmetros, começando pela propriedade *Database* onde deverá ser informado o caminho da base de dados *Employee.fdb* (<caminho>\employee.fdb). Defina as credenciais para acesso ao Firebird, usuário "SYSDBA" e senha "masterkey".

Na propriedade *ServerCharSet* utilizaremos o padrão *Win1252* e em *SQLDialect* utilize o padrão *3* para que possamos utilizar comandos de *SQL* mais complexos (**Figura 2**).

Ainda no componente *sqlConexao*, defina sua propriedade *LoginPrompt* para *False* e altere a propriedade *Connected* para *True* testando assim o sucesso da conexão.

Posicionado sobre a paleta *dbExpress*, adicione agora um componente *SQLDataSet("sqldsEmployee")* e ligue a propriedade *SQLConnection* ao componente de conexão *sqlConexao*. Adicione o seguinte comando *SQL* na propriedade *CommandText* do componente:

```
SELECT
  EMPLOYEE.FULL_NAME, EMPLOYEE.PHONE_EXT,
  DEPARTMENT.DEPARTMENT, EMPLOYEE.SALARY
  FROM EMPLOYEE
INNER JOIN DEPARTMENT
  ON (EMPLOYEE.DEPT_NO = DEPARTMENT.DEPT_NO)
```

Posicionando-se agora sobre a paleta *Data Access* adicione um componente *DataSetProvider*("dspEmployee") e relacione a propriedade *DataSet* ao *sqldsEmployee*. Ainda da paleta *Data Access* adicione um componente *ClientDataSet*("cdsEmployee"), ligando-o ao *dspEmployee* através da propriedade *ProviderName*. Altere ainda sua propriedade *Active* para *True* e com um duplo clique sobre o componente abra o *Fields Editor* e adicione todos os campos gerados pelo código *SQL* através da opção *Add All Fields* disponível ao clique do botão direito do mouse sobre o editor.

Nota: Uma boa prática de programação e economia de tempo é selecionar todos os campos do *Fields Editor* individualmente e alterar a propriedade *DisplayLabel* de acordo com o nome do campo. Dessa forma sempre que usarmos tais campos os mesmos já terão um nome amigável.

Volte agora ao formulário principal do sistema onde criaremos uma estrutura simples para a exibição dos registros e a chamada para o relatório. Adicione ao formulário um componente *DataSource*("dsEmployee"). Pressione simultaneamente as teclas *Alt + F11* e na tela que se abre escolha o *Data Module* da aplicação para que o Delphi inclua a *Unit* dele ao *Uses* do formulário.

Ligue a propriedade *DataSet* do componente ao *cdsEmployee*. Arraste também para o *formulário* um componente *DBGrid* da paleta *Data Controls* e ligue-o ao *dsEmployee* pela propriedade *DataSource*. Arraste ainda um componente *BitBtn* alterando sua propriedade *Caption* para "Imprimir em Rave Reports" e na sua propriedade *Glyph* localize um ícone para o botão, deixando a aparência do seu formulário semelhante à **Figura 3.**

Figura 1. Criando uma nova conexão

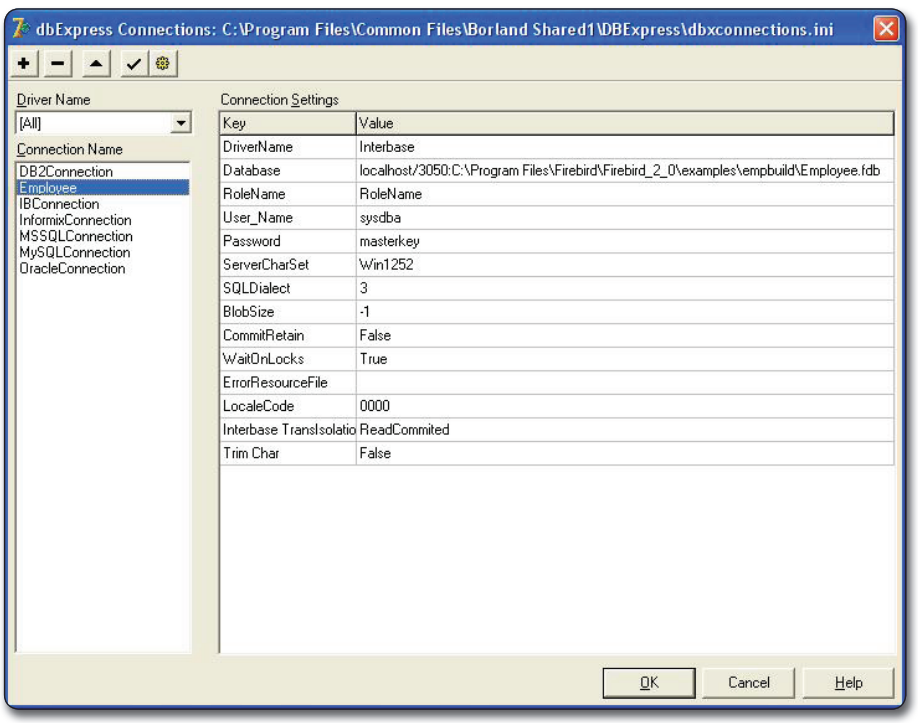


Figura 2. Configurando a conexão

No evento *OnShow* do *formulário* codificaremos de acordo com o código a seguir ativando então a busca dos registros a serem impressos:

```
DBGrid1.DataSource.DataSet.Open;
```

Compile e execute sua aplicação, de modo que poderá visualizar todos os registros atingidos pela busca. Volte à aplicação e adicione ao *Data Module* o componente *RvSystem*("rvSistema") da paleta *Rave* com a *SystemSetups>ssAllowSetup* configurada como *False* e *SystemPreview>FormState* igual a *wsMaximized*, que são responsáveis por ocultar o diálogo de *preview* antes do relatório e por abrir o relatório maximizado, respectivamente. Adicione ainda da paleta *Rave* um componente *RvProject*("rvProjeto") ligue a propriedade *Engine* ao *rvSistema*.

Em seguida adicione um *RvDataSetConnection*("rvdscEmployee") que fará a comunicação entre os dados contidos no *cdsEmployee* da aplicação e o projeto que criaremos no *Rave Reports*. Ligue sua propriedade *DataSet* ao *cdsEmployee*. Com um duplo clique sobre o componente *rvProjeto* abra o editor visual do *Rave Reports*, onde criaremos nossa estrutura de relatório.

Criando o relatório no Rave Reports

Na estrutura fornecida pelo Rave Reports, visualize na paleta ao lado direito do programa uma árvore onde o primeiro item, *Rave Project*, dá origem a vários outros, e dentre eles a *Report Library*.

O item *Rave Library* contém todos nossos relatórios e por padrão o Rave já nos oferece e exibe uma página em branco, apresentada como *Report1*. Selecionando então a página de relatório, acesse, através do *Object Inspector*, sua propriedade *Name* e altere-a para "rptEmployee", nome este que será utilizado pelo Delphi para referenciar o relatório a ser exibido. Digite o valor "Relatório de Empregados" para a propriedade *FullName*. Acesse a propriedade *Parameters* e no *Memo* de configuração digite "Solicitante", que será um parâmetro que informaremos no Delphi e que será exibido no RAVE.

Ainda nesta mesma estrutura (**Figura 4**) ao abrir os nós da *árvore* que compõem o *rptEmployee*, selecione a página do relatório (*Page1*) e altere no *Object Inspector* a propriedade *PaperSize* para "A4 Sheet, 210- by 297-millimeters", ou qualquer outro formato de papel que desejar usar na impressão do relatório. Salve o projeto do relatório no mesmo diretório onde estão armazenados os fontes da aplicação Delphi, utilize o nome "prjRave.rav".

Após adaptar o formato do papel que o relatório será impresso, iremos definir o tamanho da área imprimível e para isso iremos adicionar da paleta *Report* do Rave o componente *Region* e definir seu tamanho de modo que ocupe todo o espaço da área em branco, respeitando somente um espaçamento de cerca de 1 cm em todas as margens para que em uma impressão não seja cortada parte do texto.

Antes de começarmos a definir a estrutura dos textos do relatório, iremos adicionar ao projeto a ligação com os componentes da aplicação do Delphi.

Acessando através do menu *File|New Data Object*, selecione a opção *Direct Data View*, utilizando o botão *Next* avance para a próxima etapa onde serão listados todos os componentes de ligação disponíveis na nossa aplicação Delphi.

Selecione o *rvdscEmployee* e finalize a operação. Temos agora dentro do grupo *Data View Dictionary* na *árvore* do projeto de relatório um novo componente (*DataView1*) que possui dentro dele todos os campos que adicionamos ao *ClientDataSet* no projeto.

Com o componente *DataView1* selecionado, acesse sua propriedade *Name* e altere-a para "dvEmployee". Agora sim, voltando a acessar a paleta *Report* adicione dentro do *Region* um *Band Component*, que ficará posicionado ao topo. Selecionando a *Band1* aumente sua altura e adicione da paleta *Standard* um *Text Component* e altere seu *Text* para "Relatório de Empregados", ajuste também o tamanho da fonte para que este seja o título do nosso relatório.

Agora da paleta *Report*, logo abaixo do título adicione um *Data Text component*, acesse a propriedade *DataField* e abra o *wizard* de configuração (**Figura 5**) e digite o seguinte texto no *Memo* do grupo *Data text:*

```
'Impresso em '+Report.DateShort+' as
'+Report.TimeShort+' por +Param.Solicitante
```

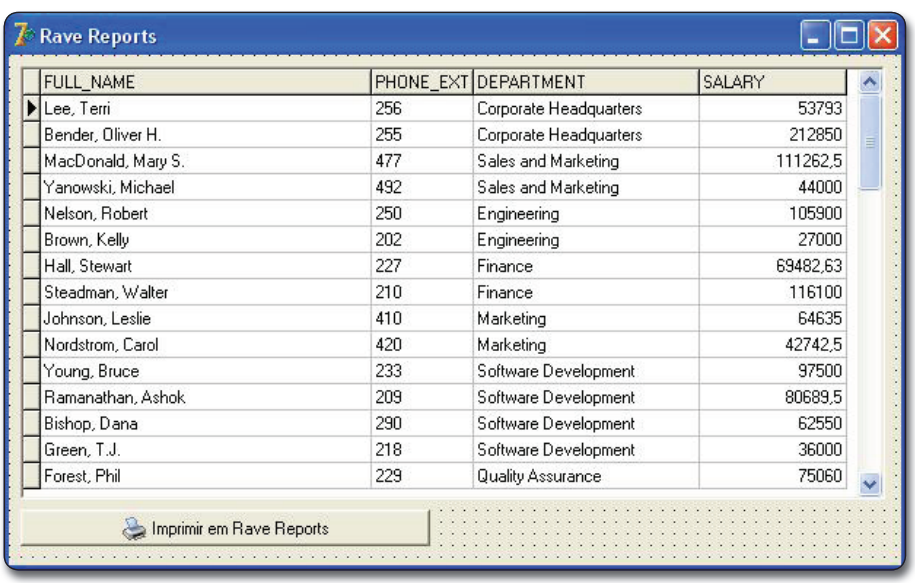


Figura 3. Estrutura do formulário principal do sistema

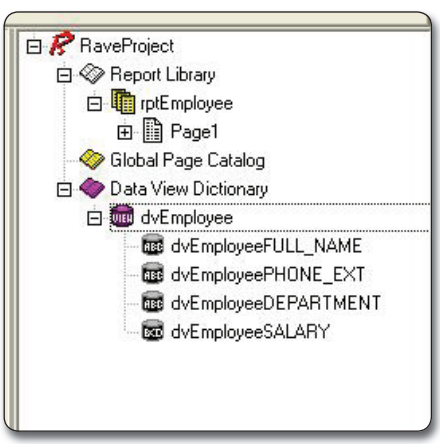
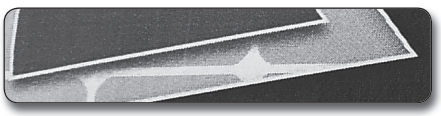


Figura 4. Estrutura do projeto no Rave Reports



Veja que no código adicionado estamos concatenando textos às funcionalidades já pré-definidas do projeto. Para incorporar os textos basta utilizar aspas simples e informar os valores entre as mesmas. Na concatenação das *strings* usa-se o sinal de adição "+". Neste caso estamos adicionando ao *DataField* a data e a hora que são capturadas do sistema operacional, além da utilização do parâmetro "Solicitante" que criamos a pouco e que será informado na codificação do Delphi com o nome da suposta pessoa "logada" no sistema.

Montando agora a estrutura onde iremos configurar os campos para a exibição dos registros do relatório.

Adicione da paleta *Reports* uma *DataBand component*, que será posicionada automaticamente também ao topo do *Region* e imediatamente abaixo do componente *Band1*. Para agilizar os serviços de configuração do relatório, iremos simplesmente arrastar os componentes que desejamos exibir para dentro da *DataBand*. Como já deixamos previamente configurados a descrição dos campos do *ClientDataSet* por meio da propriedade *DisplayLabel* de cada campo adicionado, basta que através do *dvEmployee* o campo seja selecionado e com a tecla *"Alt"* do teclado pressionada o mesmo seja arrastado para dentro da área da *DataBand*, repetindo assim o processo para todos os campos existentes.

Ao contrário da utilização da tecla *"Alt"* que adiciona somente os *Labels* de cada *Field*, ao selecionarmos novamente o campo do *dvEmployee* e manter pressionada desta vez a tecla *"Ctrl"* e arrastarmos o campo para a área da *DataBand*, será adicionado o *DataField* propriamente dito, onde em tempo de execução o mesmo irá assumir os valores a ele correspondentes.

Selecionando ainda a *DataBand component* configure sua propriedade *DataView* relacionando-a ao *dvEmployee* que referencia o local de origem das informações que nela serão exibidas.

Zebrando o relatório

Para deixar um relatório do Rave zebrado uma das formas mais simples é a utilização de um *Rectangle component* da paleta *Drawing*. Para utilizar o recurso, iremos adicionar ao *DataBand* um retângulo de modo que cubra toda a área utilizada pelos campos onde o relatório será exibido, não deixando visíveis neste momento os campos anteriormente arrastados.

Ainda com o retângulo selecionado, acesse sua propriedade *BorderStyle* alterando-a para "psClear" deixando dessa forma o componente sem margens.

Com o botão direito do mouse, envie o componente para trás dos demais através da opção *Order>Send to back*, retornando assim os campos à visibilidade. Ainda com o retângulo selecionado perceba na parte superior do espaço do relatório a existência de duas guias *Page Designer* e *Event Editor*. Selecione a guia *Event Editor* e uma nova área de propriedades e edições do relatório será exibida.

De acordo com o componente selecionado na área de desenho do relatório, diferentes eventos estarão disponíveis na opção *Defined Events*. Iremos selecionar para o nosso exemplo a opção *OnBeforePrint*, que refere-se a um evento que é chamado antes da impressão do relatório. Neste espaço do relatório podemos entrar com códigos específicos através da *Rave Language*, que são compilados e assumem diversas funções no relatório. Neste caso, iremos codificar conforme a seguir:

```
if Self.FillColor = clWhite then
  Self.FillColor := clSilver;
else
  Self.FillColor := clWhite;
end if;
```

> **Nota:** Ao adicionar códigos na Rave Language tome um certo cuidado em relação aos códigos utilizados, pois em alguns casos o compilador do Rave diferencia letras maiúsculas e minúsculas podendo assim causar erros durante a exibição do relatório.

Finalizadas todas estas etapas de configuração do relatório, faça a compilação do mesmo (*File>Execute Report* ou tecla de atalho *F9*) para certificar que nenhum erro será detectado.

Destacando valores no relatório

Uma ferramenta bastante útil e muito utilizada em relatórios com informações

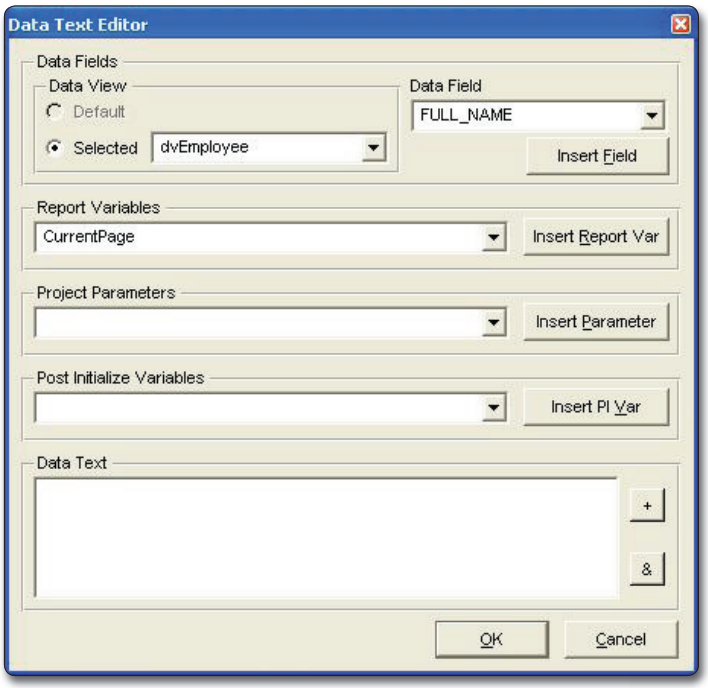


Figura 5. Wizard de configuração de um DataField no Rave Reports

de custos e valores é o destaque por cores ou fontes diferentes dos valores que atendem às condições pré-estabelecidas pelo programador ou vindas através de parâmetros da aplicação.

Iremos utilizar um bloco de código em *Rave Language* no evento *OnGetText* do *Data Text Field* responsável por exibir os registros do salário do empregado.

Selecionando o *DataText* relacionado ao *DataField SALARY*, acesse o *Event Editor* do projeto e em *Available Events* selecione o evento *OnGetText*. Digite o código a seguir. Utilize o botão *Compile* para verificar a coerência do código. Salve seu projeto e execute novamente utilizando a tecla *F9* para verificar todos os valores de salário acima de $ 100.000,00 serão apresentados em vermelho:

```
if dvEmployeeSALARY.AsFloat > 100000 then
  Self.Color:=clRed;
else
  Self.Color:=clBlack;
end if;
```

Somando valores do relatório

Utilizado em praticamente todos os relatórios que trabalham com informações de valor numérico, a soma dos campos ao final do relatório é uma informação muito útil ao usuário em análise das informações. Para adicionar o campo de soma no relatório, adicione novamente da paleta *Report* um *Band component*, que será posicionado imediatamente abaixo ao *DataBand* onde os registros são exibidos. Adicione ao *Band2* um *Text component* da paleta *Standard*, alterando sua propriedade *Text* para "Total de Salários" e adicione também da paleta *Report* um *Calc Text component* que utilizaremos para exibir a soma dos salários.

Acessando a propriedade *Calc Type* do componente temos várias opções de operações disponíveis, utilizaremos a opção *ctSum*. Precisamos informar ainda qual o campo que vamos somar. Ligue a propriedade *DataView* do componente ao *dvEmployee* e selecione em *DataField* o campo *SALARY*. Ao executar o relatório novamente, localize na última página do relatório a soma dos valores de todos os salários. Ao final de todas as configurações realizadas no projeto do relatório, a estrutura deverá ser semelhante à representada na **Figura 6.**

Executando o relatório

Retornando ao nosso projeto no Delphi, iremos adicionar os componentes para

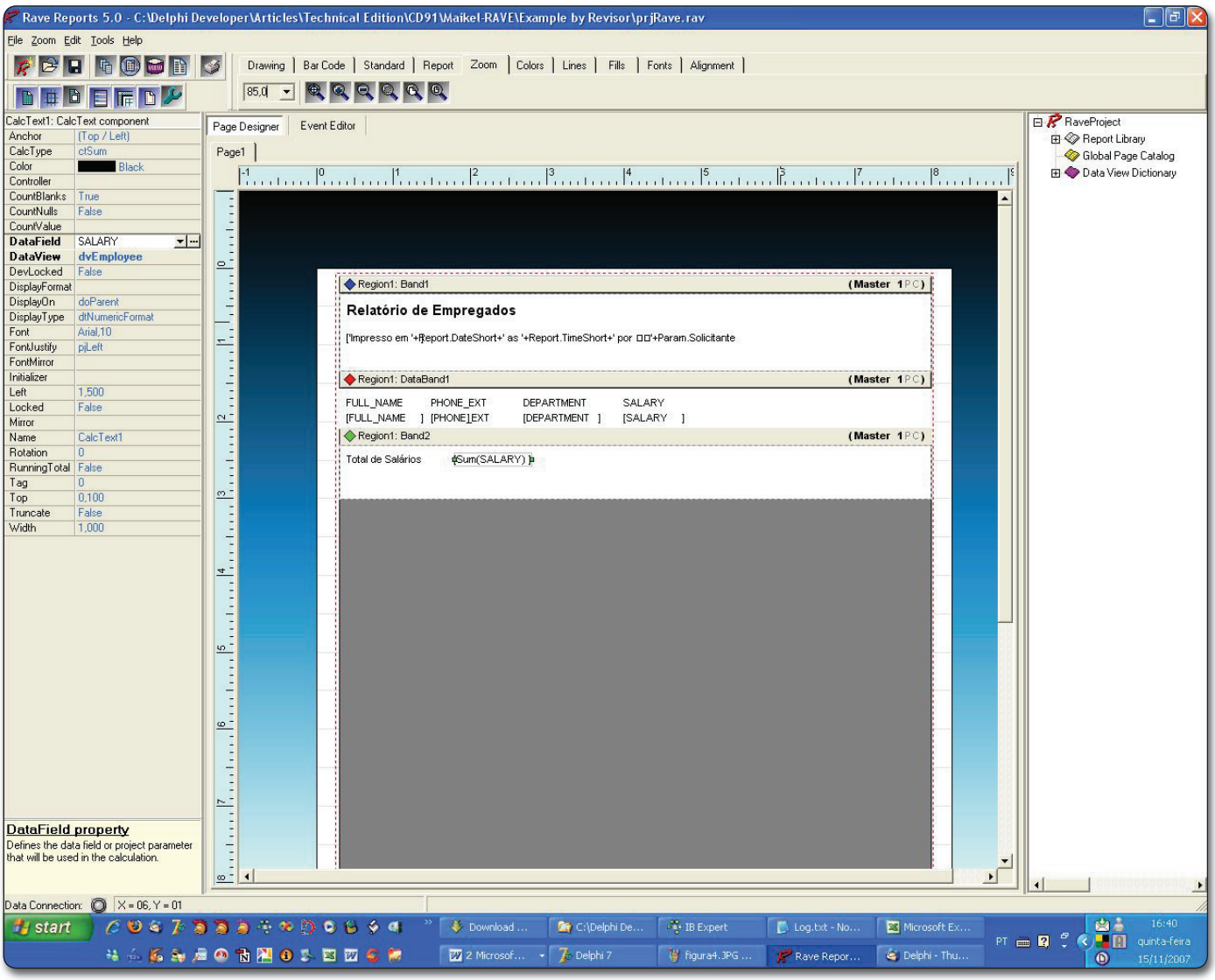


Figura 6. Estrutura do projeto de relatório no Rave

execução do relatório. Posicionando-se no *Data Module*, selecione o *rvProjeto* e na sua propriedade *ProjectFile* localize o arquivo "prjRave.rav" que será executado pela nossa aplicação. Voltando ao formulário principal do sistema, faremos a codificação para que o relatório seja exibido.

Adicione ao evento *OnClick* do *BitBtn1* as instruções do código a seguir que irá selecionar um relatório a ser impresso, passar o valor ao parâmetro que criamos no Rave e executar o relatório:

```
DM.rvProjeto.SelectReport('rptEmployee',True);
DM.rvProjeto.SetParam('Solicitante','Maikel');
DM.rvProjeto.Execute;
```

Na primeira linha do código estamos selecionando qual o relatório que queremos exibir na execução do botão. Na condição *SetParam* estamos atribuindo um valor ao parâmetro criado no relatório, informamos primeiro o nome do parâmetro e após a *string* com o valor que o mesmo deverá assumir e por último executamos e visualizamos. (**Figura 7).**

Salvando o relatório em arquivo

Uma opção interessante e que atrai muito a atenção dos clientes na hora de demonstrar o sistema é a possibilidade de salvar um relatório no formato .pdf, que poderá ser enviado pra qualquer cliente impossibilitando alterações no formato do arquivo. Para utilizar este recurso, adicione ao *Data Module* um componente *RvRenderPDF* que será o responsável por exportar o relatório para PDF.

Retornando ao formulário principal do sistema, adicione um componente *BitBtn*, configure a propriedade *Caption* para "Salvar PDF". Adicione também um componente *SaveDialog* da paleta *Dialogs* configurando sua propriedade *Filter* para "*.pdf". Adicione agora ao evento *Onclick* do *BitBtn2* o código da **Listagem 1**, que irá abrir um diálogo solicitando ao usuário o nome e a pasta que o arquivo salvou.

Compile sua aplicação e faça a exportação do seu relatório para PDF, e veja que a estrutura do relatório não foi alterada, mudando apenas o formato em que o mesmo foi salvo.

Conclusão

Vimos neste artigo dicas práticas para criação e customização dos relatórios utilizando o editor de relatórios Rave Reports. Criamos também um item prático da exportação de relatórios no formato PDF que poderá ser facilmente enviado por meio eletrônico, onde além desta possibilidade você poderá criar exportações customizadas dos relatórios ainda nos formatos *HTML, RTF e TXT.*

Além dos métodos que utilizamos no artigo você poderá criar várias outras formas de relatórios utilizando o Rave Reports, definir seu próprio *Preview* customizado e ferramentas a serem exibidas no relatório. Crie processos dinâmicos de *master/detail* utilizando quebras de grupos ou condições de exibição. ●

> **Nota:** Você poderá procurar por matérias/dicas no site do fabricante da ferramenta (em inglês) no endereço www.nevrona.com na seção *Products|Rave Reports>Tips & Tricks.*

Listagem 1. Salvando relatório em PDF

```
procedure TfrmPrincipal.Button1Click(Sender: TObject);
begin
  if SaveDialog1.Execute then
  begin
    DM.rvSistema.DoNativeOutput := False;
    DM.rvSistema.DefaultDest := rdFile;
    DM.rvSistema.RenderObject := DM.RvRenderPDF1;
    DM.rvSistema.OutputFileName := SaveDialog1.FileName;
    DM.rvProjeto.SelectReport('rptEmployee',True);
    DM.rvProjeto.SetParam('Solicitante','Maikel');
    DM.rvProjeto.Execute;
  end;
end;
```

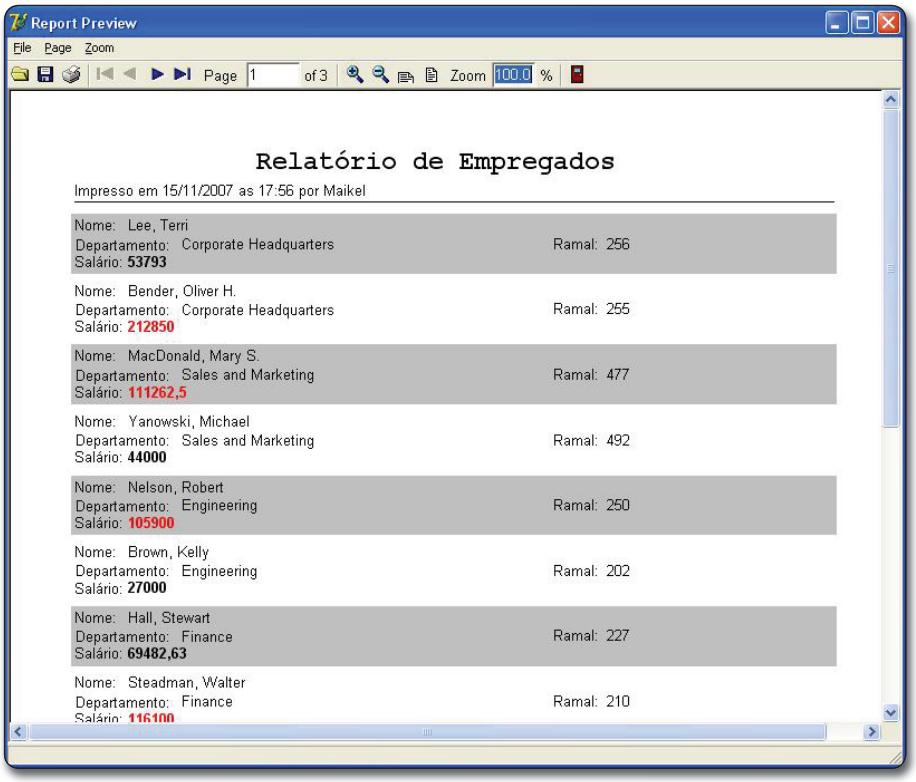
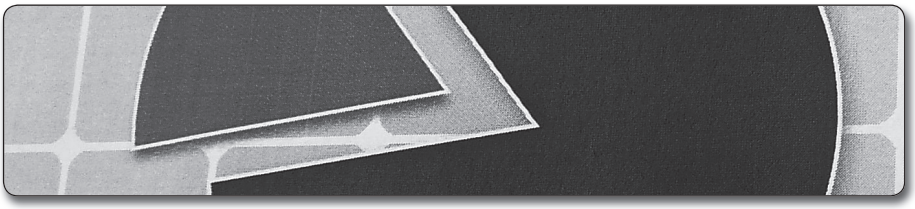


Figura 7. Exibição do relatório na tela



Nesta seção você encontra artigos para iniciantes na linguagem Delphi

Introdução ao Firebird e IBExpert

Veja como instalar o servidor e gerenciar bancos

Maikel Marcelo Scheid
(maikelscheid@gmail.com)
é técnico em Informática com ênfase em Análise e Programação de Sistemas. Atua na área de Desenvolvimento de Softwares em Delphi para plataforma Win32 e .NET com banco de dados Firebird e MS SQL. É membro da Equipe Editorial ClubeDelphi.

Indispensável a todos os sistemas de pesquisas e cadastros, o banco de dados é o responsável por manter e organizar todas as informações recolhidas e geradas por um sistema. Em sua estrutura, o banco de dados mantém organizada uma série de conjuntos de informações, tais como: listas de produtos, detalhes de compras, vendas, informações de clientes entre inúmeras outras situações onde o banco de dados mantém os registros armazenados, possibilitando futuras consultas para o resgate das informações salvas.

Neste artigo da seção *Easy Delphi* veremos a instalação do SGBD (Sistema Gerenciador de Banco de Dados) Firebird 2.0 e também a instalação do IBExpert, uma das ferramentas mais poderosas e mais utilizadas para o gerenciamento de banco de dados Interbase/Firebird.

Finalizadas as instalações criaremos uma nova base de dados através do IBExpert onde iremos definir uma estrutura de tabelas, campos e relacionamentos para armazenagem de dados. Ao final veremos a utilização dos principais comandos de linguagem *SQL* com significado mais adiante, para consultas, alterações, exclusão e edição dos registros armazenados na estrutura.

Firebird: Conceito e instalação

O Firebird possui características robustas, é leve e poderoso ao mesmo tempo, sendo assim é capaz de atender ao mesmo tempo aplicações de um usuário único ou em uma rede de múltiplos acessos em nível corporativo.

Um único servidor Firebird poderá gerenciar várias bases de dados simultaneamente com múltiplas conexões, é um SGBD de necessidades mínimas de configuração e administração, é Open Source, ou seja, livre de licenças mesmo para uso comercial, permite a criação de procedimentos (Stored Procedures) que são processados internamente em sua

camada servidora, criação de Triggers múltiplos por tabela, disparados para cada linha da tabela antes ou depois de Inserts, Updates ou Deletes com o intuito de gerar exceções, aplicar valores padrões e assegurar da integridade da base de dados.

Outra característica importante do Firebird são os *Generators/Sequences* (Geradores) que permitem colunas do tipo auto-incremento, que numeram em ordem seqüencial e crescente as linhas da tabela, além de *Backups Online*, *Integridade Referencial* em *Cascata* com controle total de suas transações. É também um SGBD multi-plataforma (*Windows, Linux, Mac OS X, Solaris e HP-UX*) com exigências mínimas de *hardware* para a instalação do servidor, que poderá ser nas versões *Classic Server* (uma instância por conexão cliente), *SuperServer* (todas as conexões são manipuladas por um único módulo) e *Embedded Version* (versão embutida onde o sistema inteiro é uma única *DLL* para uso por um único usuário e uma única base de dados).

A instalação do Firebird é bastante simples, você poderá fazer o download do arquivo executável da instalação pelo endereço www.firebirdsql.org. Veremos neste artigo a instalação da versão 2.0 do SGBD, mas que poderá ser instalado também na sua versão 1.5. A instalação propriamente dita é toda em português com várias dicas e instruções durante o processo de instalação que ocorre em 12 etapas enumeradas a seguir:

- Etapa 1: Seleção do idioma da instalação; escolha a opção Português e confirme;
- Etapa 2: Apresentação da ferramenta no assistente de instalação, utilize o botão Seguinte para continuar a instalação;
- Etapa 3: Termo de Licença; após a leitura do termo, escolha a opção Aceito o Contrato e clique em Seguinte;
- Etapa 4: Informações sobre o Firebird, dicas e configurações da instalação e remoção;
- Etapa 5: Destino da instalação, que será instalada por default no caminho "C:\Arquivos de Programas\Firebird\Firebird_2_0\", aconselhável de ser mantido;
- Etapa 6: Componentes a ser instalados; mantenha as opções padrões da instalação (**Figura 1**);
- Etapa 7: Criação de atalhos no menu/desktop;
- Etapa 8: Tarefas adicionadas a serem executadas; mantenha as opções default (**Figura 2**);
- Etapa 9: Resumo das configurações para instalação;
- Etapa 10: Instalação do Firebird 2.0;
- Etapa 11: Resumo e informações; resultado da instalação;
- Etapa 12: Início do processo para funcionamento do Firebird. Após esta etapa o serviço será iniciado automaticamente em cada restart do Sistema Operacional. Conclusão da instalação;

Neste momento o SGBD já estará instalado no computador e seus recursos poderão ser utilizados a qualquer momento para criação/hospedagem de arquivos de armazenagem de dados.

IBExpert: Conceito e instalação

O IBExpert é uma das ferramentas mais poderosas e mais utilizadas para o gerenciamento de banco de dados Firebird. Permite realizar todas as tarefas necessárias no suporte e na manutenção dos arquivos

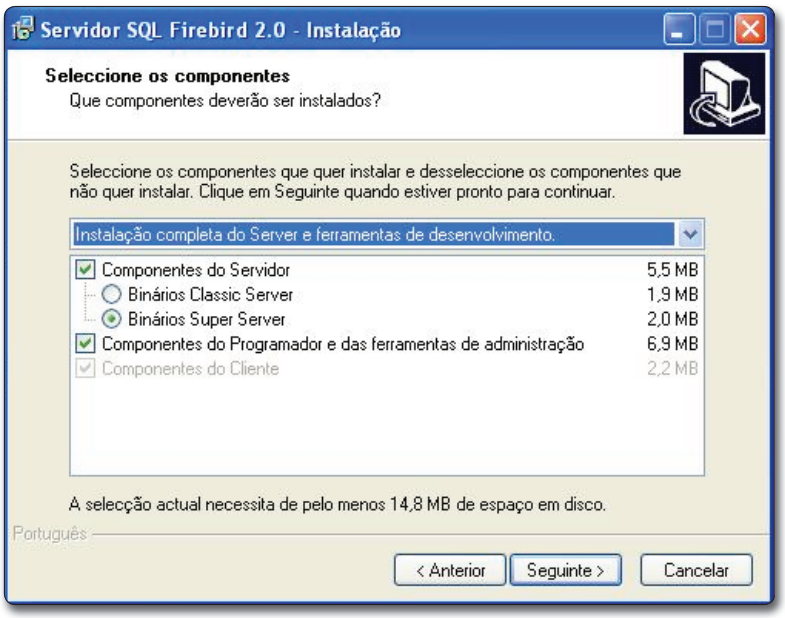


Figura 1. Componentes da instalação do Firebird

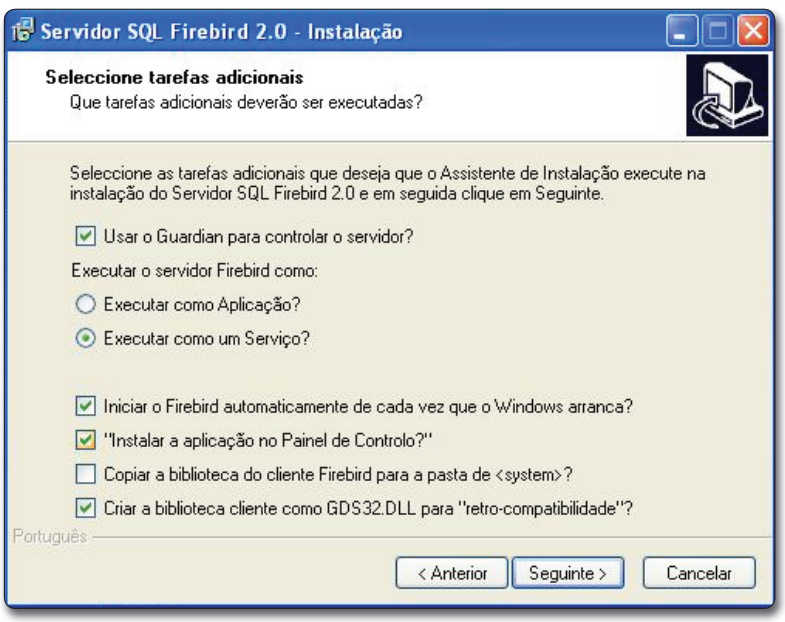


Figura 2. Tarefas adicionais a serem executadas junto ao Firebird

de banco de dados, possibilitando a criação e modificação de recursos, tabelas, campos, índices, entre outras funcionalidades, além da inserção e modificação das informações salvas na base.

Possibilita gerenciamento local ou remoto da base de dados, executando *scripts* e funções diversas. O *download* da ferramenta, que é fundamental para o *DBA* (responsável pela administração do banco de dados), poderá ser realizado pelo endereço *www.ibexpert.com*, com a possibilidade de *download* de duas versões do *software*, uma que é *free* (porém com recursos limitados) e outra versão *shareware* (com todos os recursos num período de experiência de 45 dias).

A instalação do IBExpert (em Inglês) também ocorre de forma bastante simples, num processo dividido em 6 etapas e configuração. Veremos nesse artigo a instalação da versão 2007.09.25 do produto no processo numerado a seguir:

• Etapa 1: Apresentação da ferramenta no assistente de instalação, utilize o botão Next para continuar a instalação;

• Etapa 2: Escolha dos produtos a serem instalados. Nesta tela iremos deixar marcada apenas a opção IBExpert, ignorando a instalação das demais ferramentas (**Figura 3**);

• Etapa 3: Termo de Licença; após a leitura do termo, escolha a opção I accept the agreement e clique em Next;

• Etapa 4: Pasta de destino da instalação. Mantenha o caminho default que é apresentado pelo instalador (*C:\Arquivos de Programas\HK-Software*);

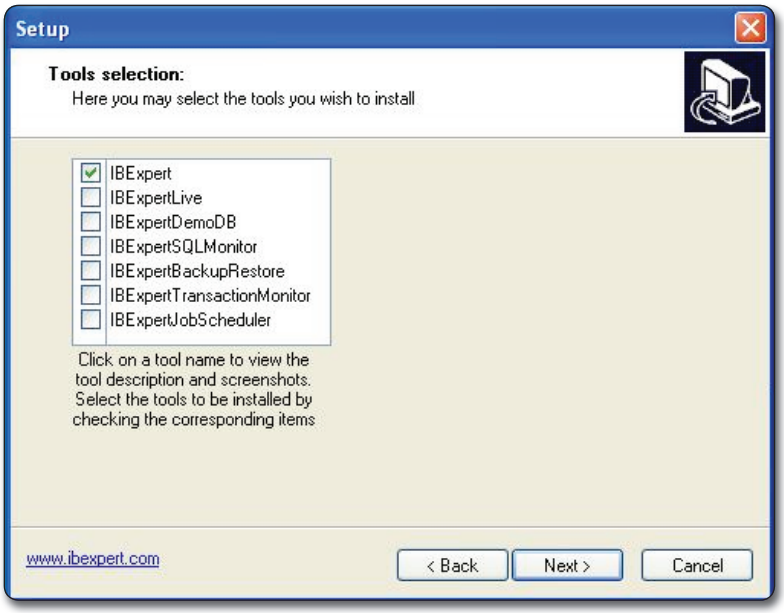


Figura 3. Configuração das ferramentas a serem instaladas

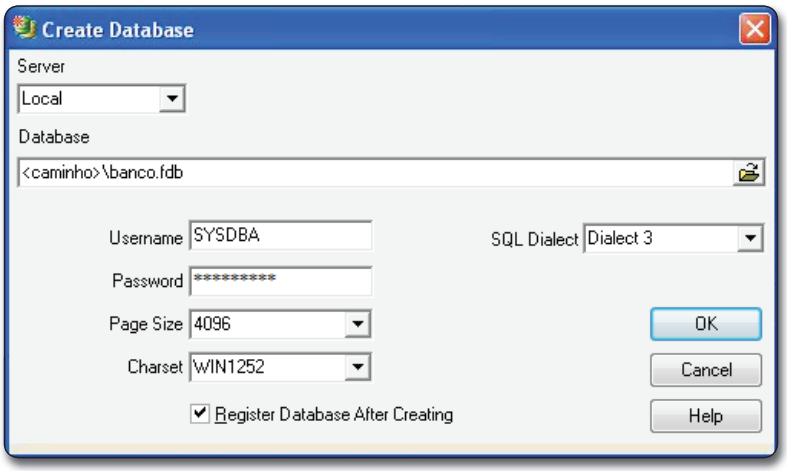


Figura 4. Criando fonte de dados

• Etapa 5: Breve resumo antes da instalação. Utilize o botão Install para continuar;

• Etapa 6: Instalação completa. Utilize o botão Finish para deixar o assistente;

Neste momento a instalação do IBExpert já está concluída e poderá ser utilizada para a criação e manutenção de arquivos de armazenagem de dados.

Criando o primeiro banco de dados

Para iniciar a criação do primeiro banco de dados, abra o IBExpert e logo após clique sobre o menu *Database>Create Database*. Uma janela será exibida (**Figura 4**) para as configurações da nova base de dados. O item Server no IBExpert oferece as opções Remote, que irá criar a fonte de dados em uma máquina da rede que está acessível ou a opção Local, usada para criar a fonte de dados nesta mesma máquina.

Na opção *Database* defina o caminho onde a fonte de dados será armazenada, atribua a fonte o nome "banco.fdb". Em relação à autenticação do usuário e senha iremos utilizar as informações específicas do Firebird informando o usuário "SYSDBA" e a senha "masterkey" que serão os mesmos para todas as instalações padrão do SGBD.

Para a opção *Page Size* (tamanho da página) utilize o valor *4096*, para *Charset* (codificação de caracteres) utilize *WIN1252*, que permitirá o uso de caracteres especiais ao inserir um registro e ainda na opção SQL Dialect (padrão de consultas e declarações SQL) defina como *Dialect* 3. Utilize o botão OK para continuar a criação da fonte de dados.

Registrando o banco de dados

Após a criação da nossa fonte de dados precisamos registrá-la no IBExpert. Se você deixou habilitada na tela anterior a opção "*Register database After Creating*" de forma automática surgirá o diálogo de registro do BD. Caso a opção tenha permanecido desmarcada, utilize o menu Database>Register Database para exibir a caixa de configurações (**Figura 5**).

No registro de um banco de dados, semelhante à criação precisamos definir o *Server* onde está o arquivo do banco, definindo então como *Local*. Ao lado temos uma configuração para definir a versão do servidor que vamos utilizar,

onde iremos definir *Firebird 2.0* que foi o SGBD instalado.

No caminho da base de dados defina o local onde o banco foi criado apontando para o arquivo "banco.fdb". Defina um nome (*Alias*) para o registro que será utilizado para identificar e diferenciar este dos demais registros de bancos que poderá fazer.

Da mesma forma que na criação da base de dados precisamos informar os usuário e senha para autenticação do registro ("SYSDBA" e "masterkey" respectivamente) e selecionar novamente a tabela de caracteres (*WIN1252*). Confirme o registro e perceba que o nome da base registrada aparecerá na paleta *Database Explorer* do IBExpert. Para conectar-se a base de dados registrada dê um clique duplo sobre o mesmo.

Nota: O IBExpert, em suas configurações, permite que seja utilizado no idioma Português. Para realizar a troca da linguagem vá até o menu *Options>Enviroment Options* e selecione a guia *Preferences*. Nela localize a opção *Interface Language*. Encontre o idioma *Portuguese* e confirme.

Criando tabelas no banco de dados

Após conectar-se ao BD, observe que estão listados vários níveis de objetos, e que dentre eles temos um nível chamado *Tables* (Tabelas). Clique com botão direito do mouse sobre *Tables* e escolha a opção *New Table* que dará origem a interface de criação e configuração da primeira tabela do banco de dados. Criaremos duas tabelas no banco, *Funcionários* e *Cargos*.

Ao acionar o comando *New Table* note que a direita do aplicativo é mostrado um campo com a palavra *New_Table*. *New_Table* é a sugestão de nome de tabela que o *IBExpert* nos dá. Então clique neste campo e digite o nome da primeira tabela que criaremos: *Funcionarios*.

Logo abaixo podemos ver uma grade com apenas um registro. Cada registro nessa base é um campo que estamos criando na tabela. Como ainda não criamos nenhum campo a indicação do IBExpert é *New_Field*. No lugar de *New_Field* digite o nome do campo e as colunas subseqüentes preencha-as de acordo com a situação.

Ex. Para o primeiro campo digite *ID_FUNCIONARIO* na coluna *Field Name*. Em *Field Type* escolha *Integer* em seguida localize a coluna *AutoInc* e marque-a.

Logo após marcar como *AutoInc* uma caixa de diálogo é aberta para configuração do auto-incremento (**Figura 6**). Na aba *Generator* marque a opção *Create Generator*. Em seguida dê um nome ao *Generator* e um valor inicial, normalmente usa-se *0*.

Para que o *Generator* funcione automaticamente usamos outro recurso do banco Firebird: *Triggers*. Então selecione a aba *Trigger* e ative a opção *Create Trigger*. As *Triggers* são também chamados de gatilhos. Esse gatilhos são disparados mediante a um evento que acontece na tabela, nesse caso *Before Insert*, ou seja, toda vez que um registro for inserido nessa tabela a *Trigger* é chamada e o *Generator* ativado.

Marque a opção *Create Trigger* e perceba que o próprio IBExpert cria a instrução SQL que será usada para incrementar o campo *ID_FUNCIONARIO* conforme a **Listagem 1**. Apenas confirme a caixa de diálogo.

Há também outra providência a se tomar. Note, nas especificações do campo ID_FUNCIONARIO, que ele é um Primary Key, ou seja, chave primária. Nesse caso para tornar um campo parte da chave primária clique na coluna PK e pressione a barra de espaços. Um ícone em forma de chave será exibido na coluna.

Nota: Na estrutura de criação de tabelas de um banco de dados não é permitido o uso de caracteres especiais. Crie todos os campos sem acentuação. Ex: FUNCIONARIO, ENDERECO, etc.

Nota: A finalidade de uma chave primária é evitar entradas de dados em redundância garantindo desta forma a integridade dos dados, como por exemplo dois ou mais funcionários com o mesmo *CPF* cadastrados no sistema. Uma *Primary Key* pode também ser composta por mais de um campo. Ex: não podemos permitir que um mesmo produto entre duas ou mais vezes na mesma nota fiscal, nesse caso os campos da chave poderiam ser *COD_NOTA* e *ITEM_NOTA* ou algo similar. Para sanar o problema, caso o usuário final realmente comprei dois produtos iguais, criamos um campo que diferencie o registro no banco de dados, como por exemplo, *Quantidade*.

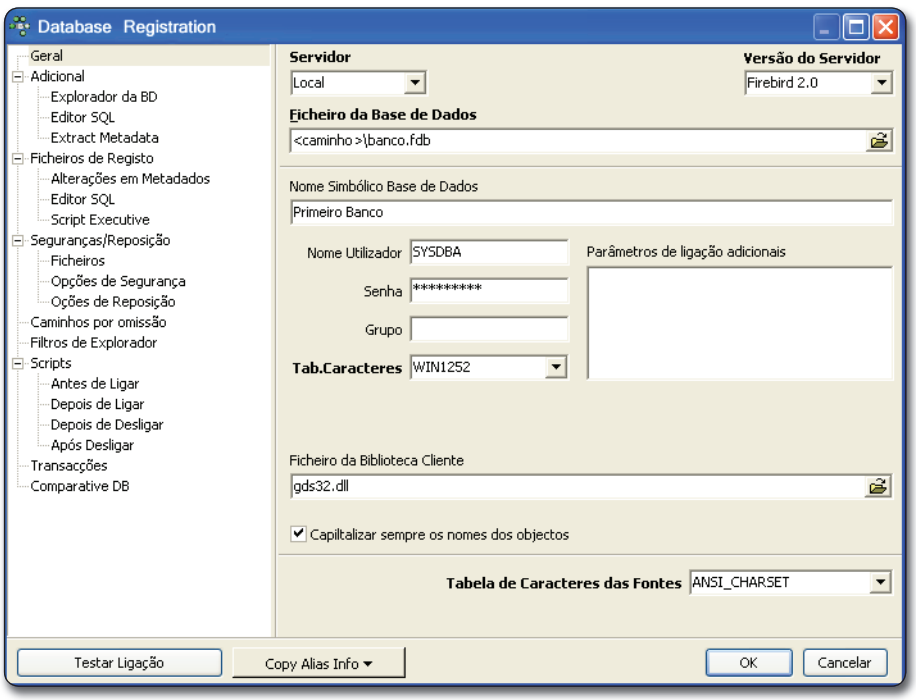


Figura 5. Registrando o banco de dados

Note que ao marcar a coluna como chave primária automaticamente a opção coluna *Not Null* é habilitada. Isso acontece porque um campo que faz parte de chave primária não pode nunca ser nulo, ou seja, precisa ser preenchido com algum valor. A estrutura completa de ambas tabelas no sistema pode ser vista em **Tabela 1** para *Funcionarios* e **Tabela 2** para *Cargos*. Crie os demais campos da sua tabela, observando apenas a alteração do tipo de dados (*Data Type*)

.

Nota: Na estrutura do campo *ID_FUNCIONARIO* marque a opção *Not Null* para o campo *ID_CARGO*, veremos a seguir como criar um relacionamento entre ambas tabelas.

Deixe sua tabela semelhante à estrutura exibida na **Figura 7.** Finalizada a criação dos campos da tabela localize no topo do aplicativo um ícone de raio com a função de compilar e criar a tabela (*Compile F9*). Na janela que irá surgir durante a compilação selecione a opção *Commit* para concretizar a inclusão.

Após criada a tabela *FUNCIONARIOS* utilize o mesmo processo para criação da tabela *CARGOS*, compile e confirme a criação da mesma.

Listagem 1. Instrução SQL da Trigger de auto-numeração do Generator

```
CREATE TRIGGER FUNCIONARIOS_BI FOR FUNCIONARIOS
ACTIVE BEFORE INSERT POSITION 0
AS
BEGIN
  IF (NEW.ID_FUNCIONARIO IS NULL) THEN
    NEW.ID_FUNCIONARIO = GEN_ID(GEN_FUNCIONARIOS_ID,1);
END
```

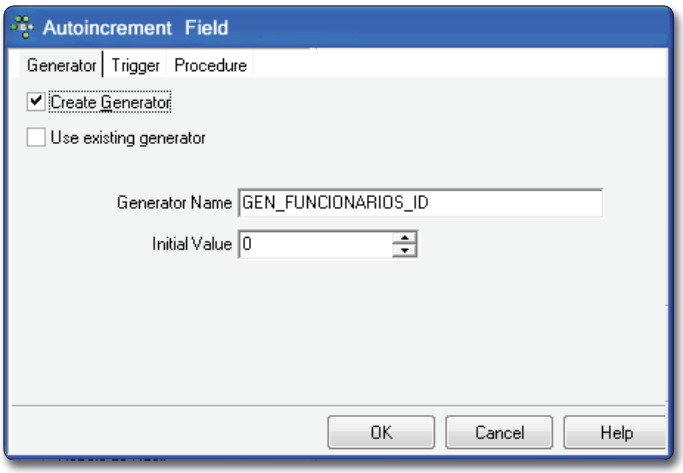


Figura 6. Criando auto-incremento na coluna ID_FUNCIONARIO

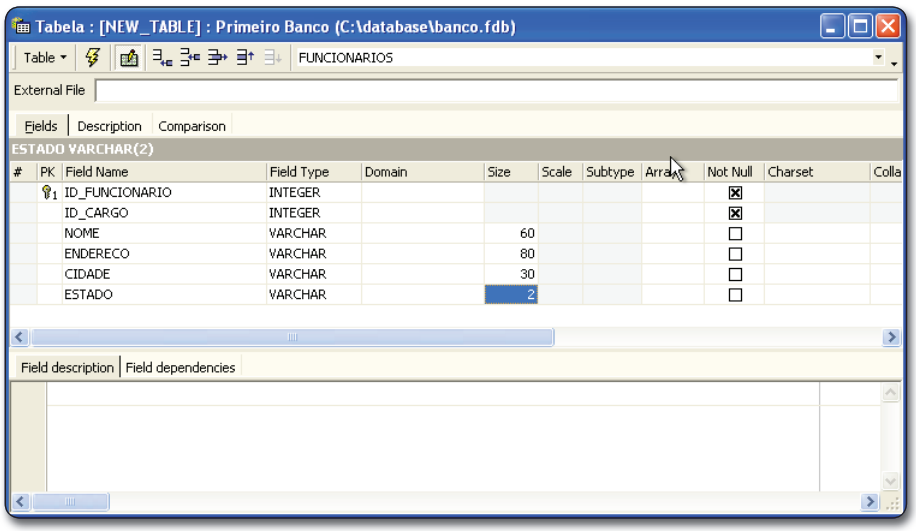


Figura 7. Tabela de FUNCIONARIOS

Criando um relacionamento, Foreign Key

Uma *Foreing Key* (Chave Estrangeira) é um relacionamento entre duas tabelas de um banco de dados. O relacionamento é criado em uma prática chamada "pai para filho" onde uma chave primária da tabela pai é referenciada por uma campo da tabela filho criando dessa forma a chave estrangeira. No nosso exemplo iremos relacionar o campo ID_CARGO da tabela CARGOS (tabela pai) com o campo ID_CARGO da tabela FUNCIONARIOS (tabela filho).

Para criar o relacionamento, abra a tabela FUNCIONARIOS com um clique duplo e observe que todos os campos estão sendo listados no IBExpert. Selecione a guia *Constraints* onde novas guias serão listadas, e entre estas localize e selecione a guia 2, *Foreign Keys*, exibindo um espaço em branco onde o relacionamento será criado.

Utilizando o botão direito do mouse sobre a área branca, selecione a opção *New foreign key*. Configurando o relacionamento mantenha o *Constraint Name* como sugerido pelo próprio *IBExpert*. Na próxima coluna, *OnField*, deverá ser informado em qual campo da tabela de *FUNCIONARIOS* vamos criar o relacionamento, selecionando o campo *ID_CARGO*. Na coluna *FK Table* localize a tabela com a qual deseja relacionar o campo

Campo	Tipo/Tamanho
ID_FUNCIONARIO	Integer(AutoInc) – Primary Key
ID_CARGO	Integer – Not Null
NOME	Varchar(60)
ENDERECO	Varchar(80)
CIDADE	Varchar(30)
ESTADO	Varchar(2)

Tabela 1. Estrutura da tabela de Funcionários

Campo	Tipo/Tamanho
ID_CARGO	Integer(AutoInc) – Primary Key
CARGO	Varchar(30)

Tabela 2. Estrutura da tabela de Cargos

selecionando, neste caso *CARGOS*, onde automaticamente na coluna *FK Field* já será listado o campo *ID_CARGO* (**Figura 8**). Utilize novamente o ícone *Compile* ou *F9* para criar o relacionamento. Para confirmar selecione o botão *Close* na janela que irá aparecer.

Ao voltar para a guia *Fields* note agora que o campo *ID_CARGO* foi marcado também com o ícone de uma chave, porém na coluna *FK*.

Inserindo/visualizando registros direto na tabela

Para inserir novos registros ou visualizá-los basta que a tabela seja exibida e sua aba *Data* esteja selecionada. Automaticamente a lista de informações já será carregada. Clicando sobre o título de uma coluna é possível fazer ordenações e pesquisa das informações cadastradas.

Abra a tabela *CARGOS*, selecione a aba *Data* e digite no espaço reservado da coluna o cargo "Programador". Ignore o campo *ID_CARGO* deixando-o sem informação, a qual será preenchida automaticamente pelo auto-incremento criado anteriormente para esta coluna. Utilize o símbolo "V" no topo da interface (*Commit Transaction*) para salvar o registro. Observe que o campo *ID_CARGO* assumiu o valor *1*. Proceda desta maneira o cadastro de mais alguns cargos.

Selecionando a tabela *FUNCIONARIOS* para inserção de registros, posicione-se na aba *Data* e clique sobre o campo *ID_CARGO*. É difícil decorarmos os códigos dos registros, mas precisamos que o código de um cargo já cadastrado seja informado nessa coluna. Para não precisarmos ir até a tabela *CARGOS* pesquisar o código do cargo que desejamos inserir basta que sejam dados dois cliques sobre a coluna e verificar que uma lista de cargos é aberta, permitindo a pesquisa por qualquer um dos campos que forem selecionados. Para salvar o registro a tabela basta selecionar o campo desejado e pressionar a tecla *Enter*. Faça o cadastro de alguns funcionários para vermos a seguir a utilização dos principais comandos de *SQL*.

Executando comandos SQL

SQL (Linguagem de Consulta Estruturada) é uma linguagem fácil e simples de pesquisa própria de banco de dados, mantendo um grande padrão entre todos os demais tipos de bancos de dados (MSSQL, MySQL, Oracle, FB etc). Veremos nessa estrutura 4 formas diferentes da utilização desta linguagem, simulando as principais ações que são realizadas em banco de dados.

Abra no IBExpert o editor de comandos *SQL* através do menu *Tools>SQL Editor* onde deverão ser digitadas as instruções de *SQL* que veremos a seguir. Para executar a instrução e observar os resultados (em caso de pesquisa) utiliza-se o ícone *Play* (*Execute – F9*). Em casos de inclusão, exclusão ou alteração de registros, além de utilizar o botão *Execute* deverá utilizar também o ícone "V" (*Commit Transaction*) para que a ação seja persistida ao banco.

1 - Consulta das informações

A pesquisa das informações cadastradas em um banco de dados é uma das transações mais utilizadas. É através deste método que conseguimos resgatar tudo o que está salvo na estrutura do mesmo. Podemos criar várias situações e formas diferentes de realizar uma pesquisa das informações, veja a seguir algumas utilizações:

Pesquisa simples: digite no *SQL Editor* a seguinte instrução de código *SQL* e execute a mesma. Estamos selecionando e exibindo todos os registros da tabela de funcionários sem qualquer restrição:

```
select * from funcionarios
```

Pesquisa parametrizada: na seguinte instrução de código *SQL* iremos definir uma condição (parâmetro) para a pesquisa, e somente os registros que atenderem a esta condição serão exibidos:

```
select * from funcionarios where
funcionarios.id_cargo = 1
```

Definimos na condição que somente os funcionários que forem pertencentes ao cargo de código número 1 serão listados, ignorando todas as demais ocorrências.

Pesquisa com relacionamentos: as pesquisas com relacionamentos são realizadas com intuito de mostrar informações de duas ou mais tabelas, utilizando as chaves estrangeiras como condições. No código da **Listagem 2** selecionamos todos os funcionários cadastrados e seus respectivos cargos. Veja que na cláusula *Inner Join* estamos definindo quem será a tabela filho (*FUNCIONARIOS*) e qual o campo que será relacionado a mesma de origem da tabela pai (*CARGOS*).

2 – Editando registros

As edições em geral são sempre de um registro específico, ou seja, um registro que atenda antes uma condição. Vejamos um exemplo prático onde iremos editar o nome de um funcionário, utilizando uma condição de que o código do mesmo deverá ser igual a *1*. Nesta edição, um único registro (*ID_FUNCIONARIO = 1*) será afetado e alterado:

```
update funcionarios set funcionarios.nome =
  'MAIKEL SCHEID'
where
  funcionarios.id_funcionario = 1
```

3 – Inserindo registros

A inserção de registros com certeza também está entre as transações mais utilizadas em bancos de dados. Sua funcionalidade é de armazenar novas informações nas tabelas. No exemplo a seguir faremos o cadastro de um novo funcionário utilizando comandos *SQL:*

```
insert into funcionarios (id_cargo, nome)
values (5,'PAULO')
```

Listagem 2. Exemplo de seleção de dados com mais de uma tabela

```
select
   funcionarios.id_funcionario, funcionarios.nome,
   funcionarios.endereco, funcionarios.cidade,
   funcionarios.estado,cargos.cargo
from
  cargos
  inner join funcionarios
  on (cargos.id_cargo = funcionarios.id_cargo)
```

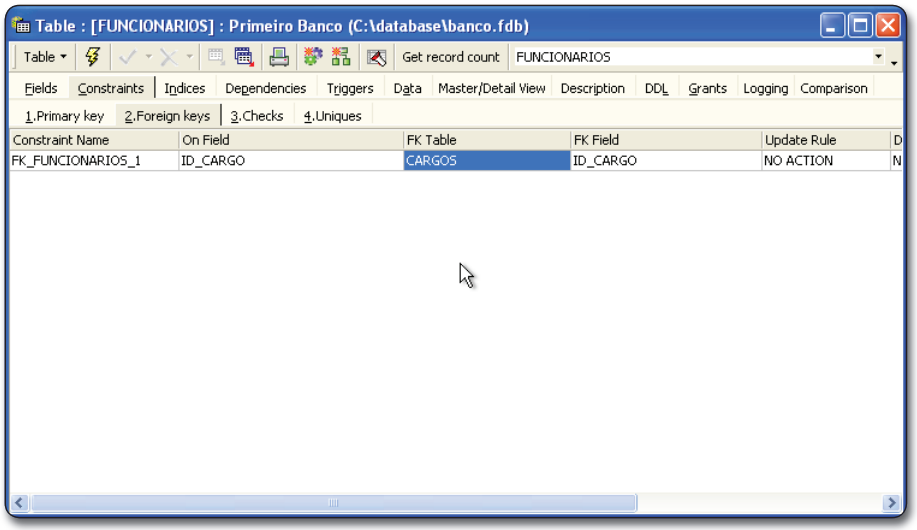


Figura 8. Criando relacionamento de chaves entre tabelas

Observe que ao passar os valores para serem inseridos na tabela de funcionários, precisamos tomar alguns cuidados em relação ao tipo de dados de cada campo. O *ID_CARGO* é de valor inteiro, podendo ser informado somente o número correspondente ao cargo, já o campo *NOME* é do tipo *Varchar* exigindo que seu valor seja passado entre aspas simples.

4 – Excluindo registros

A exclusão de registros geralmente não é muito utilizada em bancos de dados. A metodologia adotada pela grande maioria dos programadores apóia-se em não excluir as informações cadastradas, mas sim deixando-as no banco de dados como forma de histórico e atribuindo a tabela uma coluna de *STATUS*, que para cada registro irá definir se o mesmo poderá ou não ser visível ao sistema. No exemplo veremos a exclusão de um registro de funcionários, o qual deverá atender a condição de *ID_FUNCIONARIO = 6*:

```
delete from funcionarios
where funcionarios.id_funcionario = 6
```

Tome cuidado em código de exclusão para sempre utilizar um parâmetro, para não correr o risco de perder todas suas informações cadastradas.

A linguagem SQL possui uma grande variedade de comandos e requer um estudo aprofundado. O que vimos nessa seção do artigo foi apenas uma pequena introdução a linguagem. Recomendo de livros e artigos especializados em linguagem SQL para aquisição de mais conhecimento.

Conclusão

Vimos neste artigo a instalação, configuração e uso do SGBD Firebird e do gerenciador IBExpert, o qual foi utilizado para criação de um banco de dados e estudo de algumas rotinas de linguagem *SQL*.

Embora breve, a introdução à linguagem SQL poderá lhe dar um rumo para aperfeiçoar ainda mais seus conhecimentos, pesquisar novos métodos, uso de parâmetros, *procedures*, implementação de domínios nos campos de tabelas entre outras várias situações que você irá afrontar ao usar um banco de dados para criação de um sistema. Um grande abraço e até a próxima. ●

Cursos Online

Assinatura

ClubeDelphi PLUS

Mais conteúdo DELPHI por muito menos!

A Revista **ClubeDelphi** oferece para seus assinantes uma série de **Cursos Online** de alto padrão de qualidade . Conheça abaixo os cursos já disponíveis..

Curso em destaque

Aplicações Client/Server com dbExpress e Firebird

Confira neste curso online como criar uma aplicação client/server completa no Delphi 7, utilizando dbExpress e Firebird.
Aprenda também: Como trabalhar com um driver dbExpress específico e gratuito para o Firebird ; Como utilizar Orientação a Objetos, e técnicas como herança visual de formulários e relatórios ; saiba como colocar regras de negócios no banco de dados, usando Triggers e Stored Procedures; E crie relatórios com o Quick Report, Rave Reports e Report Builder, e muito mais.

Confira o plano de aula completo:
www.devmedia.com.br/clienteserver

A sua melhor opção de aprendizagem!
Assine a **ClubeDeplhi** e comece já seu treinamento!
www.devmedia.com.br/assine

Outros cursos disponíveis: **www.devmedia.com.br/curso**

- Curso Online - Sistema SysCash
- Aplicações client/server com Windows Forms no Delphi 2006
- Criando uma Aplicação multi-camadas Completa com Delphi
- Aplicações WEB com IntraWeb e Delphi 7 (Delphi Win32)

Nesta seção você encontra artigos para iniciantes na linguagem Delphi

Introdução à POO

Primeiros passos para entender os principais conceitos de Programação Orientada a Objetos

Adriano Santos
(falecom@adrianosantos.pro.br)
é desenvolvedor Delphi desde 1998. Professor e programador PHP. Bacharel em Comunicação Social pela Universidade Cruzeiro do Sul, SP. É Editor Técnico, Colunista e Membro da Comissão Editorial da revista ClubeDelphi. Mantém o blog Delphi to Delphi (www.delphitodelphi.blogspot.com) com dicas, informações e tudo sobre desenvolvimento Delphi.

POO – Programação Orientada a Objetos, ou em inglês *Object Oriented Programming* (OOP), é, sem dúvida, um paradigma entre programadores, analistas, arquitetos de software, etc. A POO vem sendo estudada em larga escala ao longo dos tempos, tomando cada vez mais espaço em reuniões de planejamento de software e afins. O motivo disso tudo são as *facilidades de codificação, reusabilidade, organização, maior flexibilidade, baixo acoplamento, encapsulamento,* e claro, *popularidade*.

De fato os recursos da POO vêm muito a calhar em qualquer tipo de software e para qualquer área, incluindo aplicações *Web,* sejam elas escritas em PHP, ASP.NET etc. Neste artigo veremos as principais características da programação orientada a objetos, como declarar procedimentos e funções chamadas de métodos, propriedades, construtores e destrutores. Aprenderemos também a desenvolver e utilizar uma classe em uma aplicação teste.

Conceitos da POO

A primeira coisa que devemos entender na POO é: por que usá-la? Bem, vejamos um exemplo bem simples de sistema que "não" se utiliza de programação orientada a objetos, desta forma entenderemos mais facilmente seus conceitos.

Abra o Delphi 7.0 ou a versão que preferir e crie uma nova aplicação em *File|New>Application*. Nomeie o formulário principal com o nome de "frmPrincipal" e salve-o como "uPrincipal.pas" usando o menu *File|Save as*. Em seguida salve a aplicação com o nome de "ModoProcedural.dpr" através do menu *File>Save project as*.

Certamente você deve ter achado estranho o nome que salvamos o nosso projeto principal: *Modo Procedural*. Realmente é estranho, mas é mais ou menos assim que chamamos um sistema que não é orientado a objetos. Chamamos de "Programação Procedural".

"O termo Programação Procedural (ou programação procedimental) é às vezes utilizado

como sinônimo de Programação Imperativa (paradigma de programação que especifica os passos que um programa deve seguir para alcançar um estado desejado), mas o termo pode se referir (como neste artigo) a um paradigma de programação baseado no conceito de chamadas a procedimentos. Procedimentos, também conhecidos como rotinas, sub-rotinas, métodos, ou funções (que não devem ser confundidas com funções matemáticas, mas são similares àquelas usadas na programação funcional) simplesmente contêm um conjunto de passos computacionais a serem executados. Um dado procedimento pode ser chamado a qualquer hora durante a execução de um programa, inclusive por outros procedimentos ou por si mesmo".

Segundo o verbete anterior, retirado do site *www.wikipedia.org*, *programação procedural* significa que o sistema deve seguir certo fluxo, certo número de instruções também conhecidas como procedimentos. Vejamos na prática uma aplicação orientada a procedimentos.

Insira em nosso *frmPrincipal* dois componentes do tipo *SpinEdit* da paleta *Samples*. Modifique o *Name* desses componentes para "speValor1" e "speValor2", respectivamente. Em seguida adicione um *Button*("btnCalcular') da paleta *Standard*. Por último acrescente um *Label*("lblSoma") também da paleta *Standand*. Se preferir, insira outros *Labels* acima dos *SpinEdits* e modifique seus respectivos *Captions* para "Valor 1" e "Valor 2". Veja o layout sugerido para a tela na **Figura 1**.

Fiz, é claro, algumas mudanças na parte visual, basicamente modifiquei as fontes dos controles visuais, mas isso nada tem a ver com orientação a objetos. Vamos nos concentrar nos códigos. Clique duas vezes no *btnCalcular* e digite o código da **Listagem 1** em seu evento *OnClick* e vejamos o que estamos codificando.

Basicamente não há muito segredo. Declaramos três variáveis que recebem, respectivamente, o valor digitado no *SpinEdit1* e *SpinEdit2* e por fim a soma dos dois valores. Depois exibimos o resultado no *Caption* do *Label*. Veja a aplicação em execução na **Figura 2**.

Se analisarmos bem o código, veremos que o evento do botão, não somente do botão, mas todo objeto no Delphi é um procedimento, uma *procedure*.

```
procedure TfrmPrincipal.
btnCalcularClick(Sender: TObject);
```

Logo podemos dizer que nosso sistema executa um procedimento para efetuar a soma dos dois valores digitados nos *SpinEdits*. E o código para calcular a soma de ambos componentes ficou acessível somente ao clique do botão. Porém, vamos pensar em uma situação: imagine agora que queiramos incluir dois novos componentes *SpinEdit* em uma nova janela e os mesmos também poderão ser somados da mesma forma.

Vamos fazer um exemplo: inclua um novo formulário usando a opção *File|New>Form*. Salve o formulário como "uCalculo2.pas" e modifique seu *Name* para "frmCalculo2". Para facilitar selecione todos os componentes do *frmPrincipal* e cole-os no novo formulário, assim teremos a mesma estrutura que no formulário anterior. Copie o código da **Listagem 1**, ou seja, o código do evento *OnClick* do botão *btnCalcular*, para o evento *OnClick* do *btnCalcular* do segundo formulário.

Como copiamos e colamos, os nomes de componentes permaneceram os mesmos. Para que tenhamos acesso ao segundo formulário, insira um segundo *Button*("btnFormulario2') no formulário principal, modifique seu *Caption* para "Form 2" e digite o código a seguir em seu evento *OnClick*.

```
frmCalculo2.ShowModal;
```

Execute a aplicação, clique no *btnFormulario2* e simule uma soma. Claro que o resultado será o mesmo, ou seja, o sistema funcionará sem qualquer problema. Veja na **Figura 3** a aplicação novamente em execução.

Porém devemos pensar em uma hipótese: e se essas duas telas fossem janelas de entradas de dados em um sistema real? E a soma executada pelos botões *btnCalculo* de ambas telas fosse algo padronizado no sistema, como uma fórmula para cálculo de algum imposto federal?

A expressão para cálculo certamente está propensa a futuras alterações, ainda mais em se tratando de imposto federal. Agora imagine que uma determinada lei seja modificada e conseqüentemente a expressão de nosso cálculo seja modificada? Em um sistema real com poucas telas a manutenção do software seria pequena e rapidamente poderíamos alterar todas as telas que possuem o cálculo disparando

Listagem 1. Código do evento OnClick do botão btnCalcular

```
procedure TfrmPrincipal.
btnCalcularClick(Sender: TObject);
var
  Valor1 : Integer;
  Valor2 : Integer;
  Resultado: Integer;
begin
  Valor1 := speValor1.Value;
  Valor2 := speValor2.Value;
  Resultado := Valor1 + Valor2;
  lblSoma.Caption := ntToStr(Resultado);
end;
```

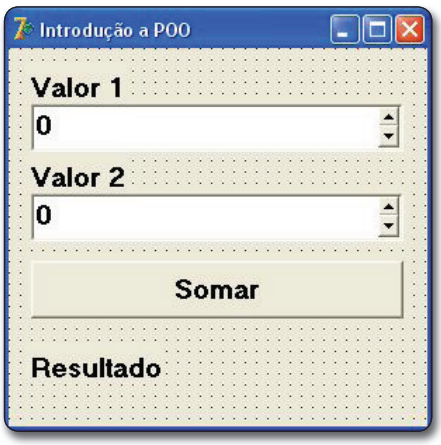


Figura 1. Exemplo de tela

Figura 2. Aplicação em execução

Figura 3. Aplicação em execução com dois formulários

mais tarde uma atualização do programa para nossos clientes. Entretanto, em um projeto com dezenas ou centenas de formulários a probabilidade do programador responsável esquecer de alterar algumas dessas telas é grande.

Em outras palavras o que estou tentando dizer é que precisamos criar artifícios para centralizarmos nossas regras de negócios (as regras da aplicação ou impostas pelo problema que se quer solucionar) em um único lugar, de forma que possamos alterá-las sempre que necessário.

Existem inúmeras formas de se resolver esse problema, uma delas é utilizando técnicas de POO.

Estudando a POO

Antes de mais nada, gostaria de deixar claro que POO é assunto para vários livros. Em segundo lugar, quando se fala em Delphi e POO, o assunto é polêmico. Por exemplo, no exemplo, anterior, mesmo sem saber acabamos usando alguns recursos da POO, como herança (o Delphi cria um novo form baseado em *TForm*). Com isso em mente, podemos prosseguir.

Antes de resolvermos nosso estudo de caso vamos entender um pouco melhor o que é POO. Podemos dizer que a POO propriamente dita é um conjunto de classes de objetos com atributos/características específicas que descrevem o software.

Mas e o que é exatamente um objeto? E o que são atributos/características? Vamos a algumas analogias para que possamos entender melhor um objeto. Ao entrarmos em uma loja de sapatos, evidentemente para comprar um sapato ou tênis, nos deparamos com uma grande variedade de calçados. Ao sermos abordados pelo vendedor certamente as primeiras perguntas que ele nos fará serão:

- Qual modelo de sapato o senhor deseja?
- Com cadarço ou sem cadarço?
- Que cor prefere?

Em um sapato o *modelo*, se tem *cadarço* ou não e a *cor* são atributos, ou seja, características do objeto. E que são particulares desse tipo de objeto. Jamais podemos dizer que um *carro* tem *cadarço* ou não, pois esse atributo não se encaixaria com o objeto.

Aprofundando-se um pouco mais, objetos podem ter métodos, que são basicamente procedimentos ou funções. Um exemplo de método para um objeto carro seria *Acelerar*. Um carro acelera, logo entra em movimento e ganha velocidade. Isso significa que se o carro entra em movimento, certamente uma hora terá que parar, ou seja, podemos dizer que um *carro* também precisa de um método *Parar*. Ambos podem ser *eventos* que são desencadeados de acordo com a situação. Exemplos de eventos *aoAcelerar*, *aoParar*, etc.

Outra parte importante da POO que devemos entender é o conceito de classes. Uma classe representa um conjunto de objetos de características semelhantes. Voltando a analogia do *sapato*, fica simples entendermos uma classe se dissermos que um *sapato* é um *calçado* e logo pode estar dentro de uma classe chamada *Calçado*.

Um *calçado* pode ter como objeto um *sapato* ou um *tênis*, ambos detêm características em comum, tais como: *solado* e *palmilha*. Porém o *tênis* não pode ser engraxado, já o *sapato* sim. Com base nesse ponto de vista podemos resumir da seguinte forma: ambos os objetos descendem de *Calçado* por terem características em comum, mas são diferentes em outros aspectos, por isso cada um pode ser um objeto distinto.

Trazendo para a realidade do exemplo

Em nosso exemplo criaremos uma classe *TCalculo* capaz de somar automaticamente os valores e nos retornar o resultado. Dessa forma teremos o código (a expressão de soma) encapsulado em uma *Unit* que poderemos usar em qualquer parte do software ou de outro projeto qualquer.

Listagem 2. Declaração da classe, seções e função

```
unit uCalculo;
interface
type
  TCalculo = class
  public
    function Somar(AValor1, AValor2: Integer): Integer;
  end;
implementation
end.
```

Listagem 3. Evento OnClick do btnCalcula agora usando classe

```
procedure TfrmPrincipal.btnCalcularClick(Sender: TObject);
var
  Valor1 : Integer;
  Valor2 : Integer;
  Resultado: Integer;
  Calculo : TCalculo;
begin
  Calculo := TCalculo.Create;
  Valor1 := speValor1.Value;
  Valor2 := speValor2.Value;
  Resultado := Calculo.Somar(Valor1, Valor2);
  lblSoma.Caption := IntToStr(Resultado);
end;
```

Listagem 4. Evento OnClick simplificado

```
procedure TfrmPrincipal.btnCalcularClick(Sender: TObject);
var
  Calculo : TCalculo;
begin
  Calculo := TCalculo.Create;
  lblSoma.Caption := IntToStr(Calculo.Somar(speValor1.Value, speValor2.Value));
end;
```

Listagem 5. Código completo da classe depois de criadas as propriedades

```
type
  TCalculo = class
  private
    FVlr1: Integer;
    FVlr2: Integer;
    FResultado: Integer;
  public
    procedure Somar;
    property Vlr1: Integer read FVlr1 write FVlr1;
    property Vlr2: Integer read FVlr2 write FVlr2;
     property Resultado: Integer read FResultado
       write FResultado;
  end;
```

Nota: Encapsular significa separar aspectos internos e externos de um objeto. Isso é bastante utilizado quando se deseja impedir o acesso a determinados elementos de um objeto. Deixando externo somente aquilo que realmente interessa. Exemplo: você não precisa conhecer profundamente as engrenagens do câmbio do carro para utilizá-lo.

Nota: Em Delphi todas as classes são iniciadas com a letra "T" como padrão da linguagem, mas é perfeitamente possível desenvolver classes sem o prefixo indicativo.

Para isso clique em *File|New>Unit* e salve-a como "uCalculo.pas". Após a palavra reservada *Interface* crie uma seção *Type* e em seguida a classe *TCalculo*, como a seguir:

```
type
  TCalculo = class
```

Logo após crie uma seção *public* e declare uma *function* (função) chamada *Somar* com os parâmetros *Valor1* e *Valor2* do tipo *Integer* e retorno também *Integer*. Não é difícil, veja na **Listagem 2** como fica o código completo das declarações.

Pressione *Ctrl + Shift + C* que automaticamente o Delphi criará o cabeçalho da função e o redirecionará diretamente para ela. Digite na função apenas o código a seguir:

```
Result := Vlr1 + Vlr2;
```

Agora podemos usar a função *Somar* em nossa aplicação. Vejamos como funciona. Retorne ao *frmPrincipal* selecione o menu *File|Use unit* para que possamos adicionar a *Unit* que contém a classe na seção *Uses* do formulário principal. Feito isso entre no código do evento *OnClick* do *btnCalcular* e altere o código para que fique conforme a **Listagem 3**.

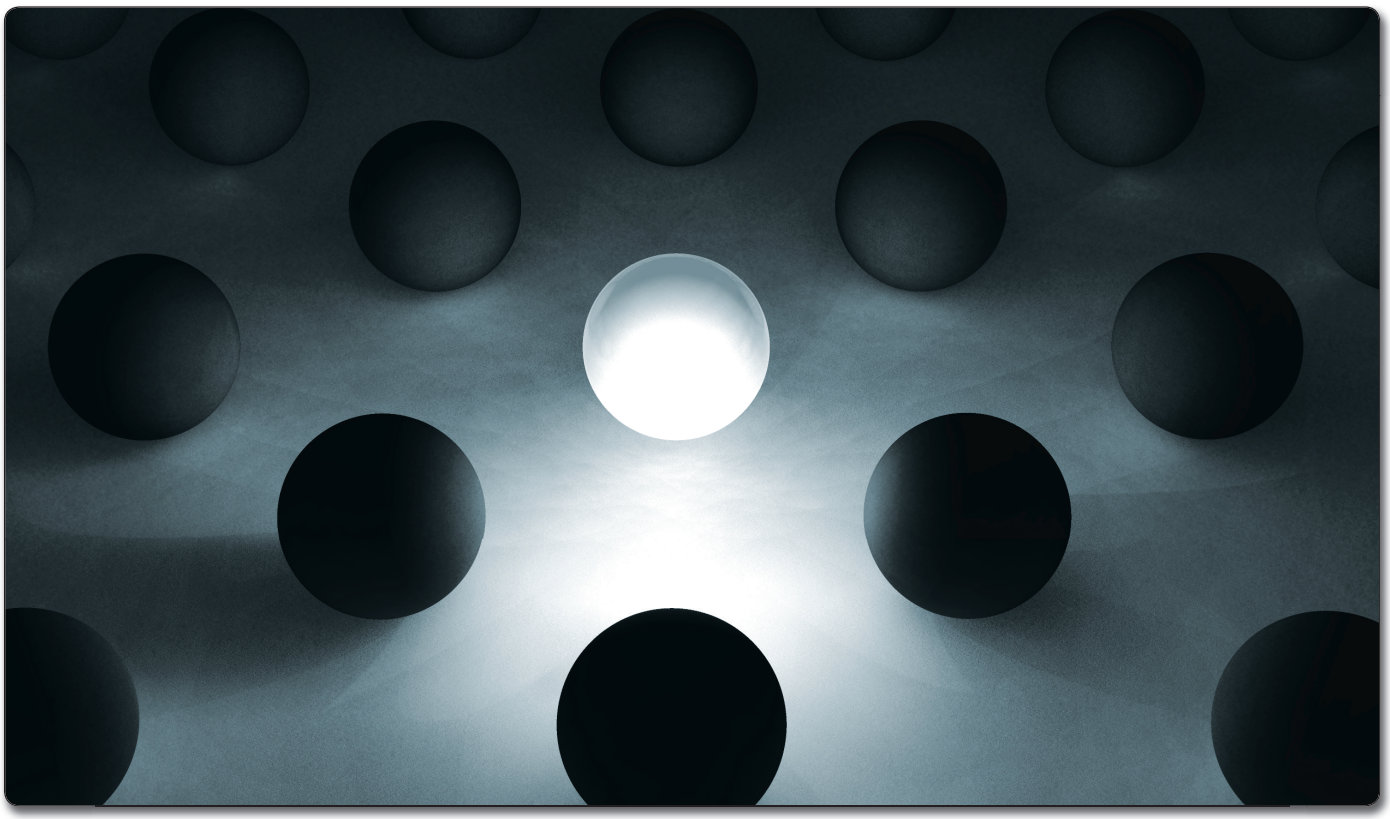
O que estamos fazendo na **Litagem 3** é criar uma variável do tipo *TCalculo*, nossa classe, e usar seu método *Somar* para efetuar a soma entre os dois *SpinEdits*. Repita os passos anteriores no *frmFormulario2* e rode o sistema. Agora se precisarmos alterar a expressão basta entrarmos no método *Somar* da classe e fazer a alteração que for necessária, desta forma todo o sistema já estará alterado automaticamente. Basta compilar novamente e distribuir aos clientes.

Pra simplificar ainda mais, vamos modificar o evento OnClick de ambos botões. Entre novamente no evento citado e altere o código conforme a **Listagem 4**.

Note que retiramos as variáveis e colocamos a função *Somar* diretamente no parâmetro da função *IntToStr*. Isso significa que o resultado de *Somar* será passado

Listagem 6. Código adaptado para trabalhar com propriedades

```
procedure TfrmCalcular2.btnCalcularClick(Sender: TObject);
var
  Calculo : TCalculo;
begin
  Calculo := TCalculo.Create;
  Calculo.Vlr1 := speValor1.Value;
  Calculo.Vlr2 := speValor2.Value;
  Calculo.Somar;
  lblSoma.Caption := IntToStr(Calculo.Resultado);
end;
```



diretamente para *IntToStr* que por sua vez converte o valor em *string* e atualiza a propriedade *Caption* do *Label*.

Trabalhando com propriedades

Faremos mais uma alteração em nossa classe. Incluiremos propriedades, ou seja, atributos da classe *TCalculo*. Para isso entre na *Unit uCalculo* onde encontra-se nossa classe e adicione abaixo da declaração de nossa *function* três propriedades como segue:

```
property Vlr1: Integer read FVlr1 write FVlr1;
property Vlr2: Integer read FVlr2 write FVlr2;
property Resultado: Integer read FResultado
 write FResultado;
```

Após digitar o código anterior pressione *Ctrl* + *Shift* + *C* e note que o Delphi cria para nós uma seção *private* e nela duas variáveis. Exclua a *function* que criamos, tanto na declaração quanto no corpo da nossa *Unit*, e declare agora uma *procedure* sem parâmetros, como segue:

```
procedure Somar;
```

Em seguida pressione novamente *Ctrl* + *Shift* + *C*. No corpo do procedimento *Somar* digite o código em seguida:

```
FResultado := FVlr1 + FVlr2;
```

Perceba agora que ao invés de usarmos *Vlr1* e *Vlr2*, estamos usando *FVlr1* e *FVlr2*. O prefixo "F" na frente da variável é padrão do Delphi e significa *Field* (campo). Como podemos ver no código da **Listagem 5**.

Um *Field* é basicamente uma variável que guarda o valor para uma propriedade. A propriedade é o atributo que fica exposto para o mundo externo, a variável *Field* não, apesar de ser ela a responsável por internamente armazenar o valor em memória atribuído a uma variável.

Agora será preciso alterar novamente o evento do botão *btnCalcular* em ambos formulários para se adaptar as novas regras da classe. Veja na **Listagem 6** como fica o novo código trabalhando com propriedades.

Vejamos o que foi feito: criamos a classe, alteramos as propriedades *Vlr1* e *Vlr2*, chamamos o procedimento (método) *Somar* e então passamos para a função *IntToStr* o conteúdo da propriedade *Resultado* da nossa classe. Note na **Figura 4** que o Delphi não exibe as variáveis *FVlr1*, *FVlr2* e *FResultado*, isso porque elas estão *encapsuladas* (ocultas) dentro da classe.

Tratando exceções na POO

Por último vamos fazer um pequeno tratamento de erro em nossa classe para evitar que o usuário final envie números negativos para a soma. Volte na *Unit* da classe e, no corpo do procedimento *Somar*, digite o código a seguir:

```
if (FVlr1 > 0) and (FVlr2 > 0) then
  FResultado := FVlr1 + FVlr2
else
  raise Exception.Create('Digite somente
    números positivos.');
```

Como não podemos e nem é recomendado utilizar diálogos em classes, então geramos uma exceção usando a palavra reservada *raise*. Estamos verificando se ambos os valores são maiores que zero, caso não sejam uma exceção é levantada e uma mensagem de erro é enviada a aplicação, conseqüentemente ao usuário final (**Figura 5**).

Com todas essas precauções temos tranqüilidade na hora de fazer uma manutenção e/ou implementação no sistema, pois organizamos o código, fizemos rotinas reutilizáveis, encapsulamos parte das regras de negócios e centralizamos as rotinas mais importantes. Como mencionado, existem diversas formas de se chegar a um mesmo objetivo. Mas com certeza a POO é uma das melhores e mais rápidas soluções em nossa atualidade.

Conclusão

A POO é fantástica e nos abre infinitas possibilidades no mundo do desenvolvimento. Com ela é possível desenvolver soluções robustas e sólidas. Recomendo o estudo amplo e dedicado às técnicas de POO. Cada vez mais a programação orientada a objetos conquista o mercado de trabalho e o profissional que dominar as técnicas terá grande futuro pela frente.

Um forte abraço e até a próxima. ●

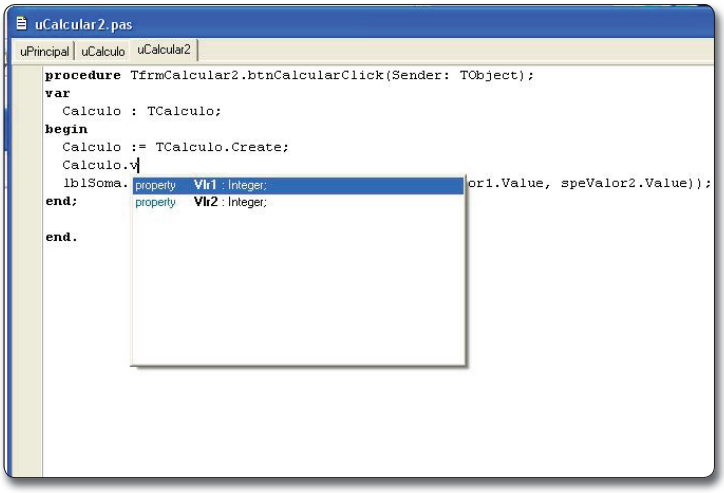


Figura 4. Não podemos alterar diretamente FVlr1 e FVlr2

Figura 5. Exceção gerada pela classe

esta seção você encontra artigos para iniciantes
a linguagem Delphi

Treinamento em ASP.NET – Parte 2

Primeiros passos com ASP.NET

Renato Haddad
é Microsoft Most Valuable Professional (MVP). Autor de diversos treinamentos multimídia .NET e SQL Reporting Services para Microsoft Brasil e América Latina.

Vamos à segunda parte do treinamento em ASP.NET, na qual você aprenderá a manipular alguns dos controles utilizados com mais freqüência durante o acesso a banco de dados. Atualmente é normal que os dados fiquem armazenados em alguma fonte de dados, seja um banco de dados no SQL Server, Oracle, MS-Access, DB2, Firebird ou então, em arquivos XML e textos. No .NET, isso é completamente transparente, ou seja, não importa onde estejam os dados.

Escolhendo o provider para ADO.NET

Temos somente uma opção para acessar o Firebird a partir de aplicações construídas para o .NET Framework: o ADO.NET. Sabendo disso, o próximo passo é escolher um *Provider* para ADO.NET que permita o melhor acesso ao banco de dados.

O .NET Framework 1.1 já é distribuído com quatro *Providers* nativos para ADO.NET:

SQL Provider - para acesso ao SQL Server;
Oracle Provider - para acesso ao Oracle;
OleDB Provider - para acesso a fontes de dados que possuam um driver OleDB;
ODBC Provider - para acesso a fontes de dados que possuam um driver ODBC;

Para acessar o Firebird, poderíamos usar qualquer um dos dois últimos *Providers* (ODBC ou OLEDB). No entanto, seria necessário instalar uma camada extra (um driver OLEDB ou ODBC), o que comprometeria o tempo de resposta, que é crítico em soluções Web.

Download e instalação do Firebird Data Provider

Acesse o endereço *www.firebirdsql.com*, clique no link *Download* e a seguir em *Firebird .NET Data Provider*. Baixe e instale a última versão do *Data Provider for .NET Framework 1.1*, bastando seguir os passos no assistente que será iniciado.

Nota: Neste artigo usaremos versão 1.7 do *Provider*, versão mais recente disponível até o fechamento desta edição.

Após a instalação, no Delphi 2006 clique no menu *Component|Installed .NET Components*. Digite "Firebird Data Provider" na opção *Category*, clique no botão *Select an Assembly* **(Figura 1)** e escolha o arquivo *FirebirdSql.Data.Firebird.dll*, localizado no diretório de instalação do *Provider*, por padrão em *C:\Arquivos de programas\FirebirdNETProvider(versão)*. Clique em *Ok* e observe que os novos componentes para acesso ao Firebird estão agora disponíveis no IDE **(Figura 2)**.

Desenvolvendo o exemplo

Na parte 1, você aprendeu a criar uma aplicação .NET no Developer Studio 2006. Então, abra a aplicação *DelphiMag*, criada na parte 1, e adicione um novo formulário usando a opção *File|New|Other>Delphi for .NET Projects>New ASP.NET Files>ASP.NET Page***).** Salve o arquivo como "BancoDados.aspx" usando a opção *File>Save as*. Vamos montar um formulário semelhante ao da **Figura 3**. Para isso a primeira providência é digitar o título principal da página. Clique em qualquer lugar do formulário e digite "Banco de Dados – ASP.NET". Para formatar basta selecionar o texto e usar os botões na barra de ferramentas superior. Em seguida pressione *Enter* e clique no botão *Insert table* também na barra de ferramentas.

Na caixa de diálogo que se abre (**Figura 4**) digite "3" para *Rows* (linhas) e "2"

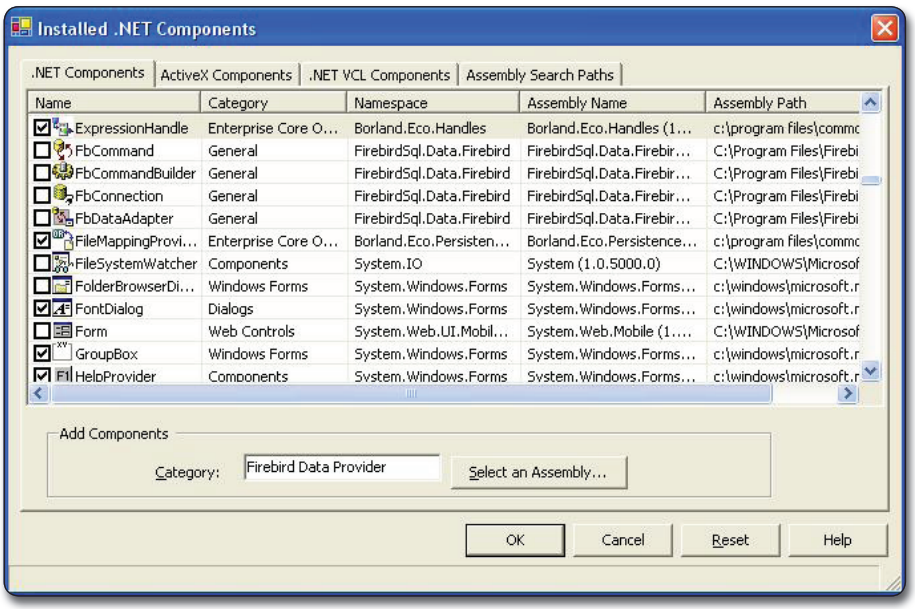


Figura 1. Instalação do provider .NET no Delphi

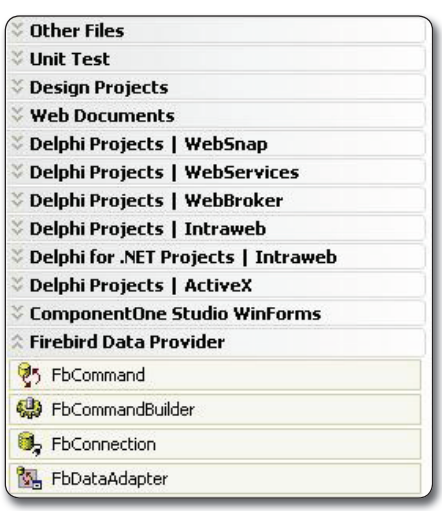


Figura 2. Componentes do provider .NET Firebird já instalados

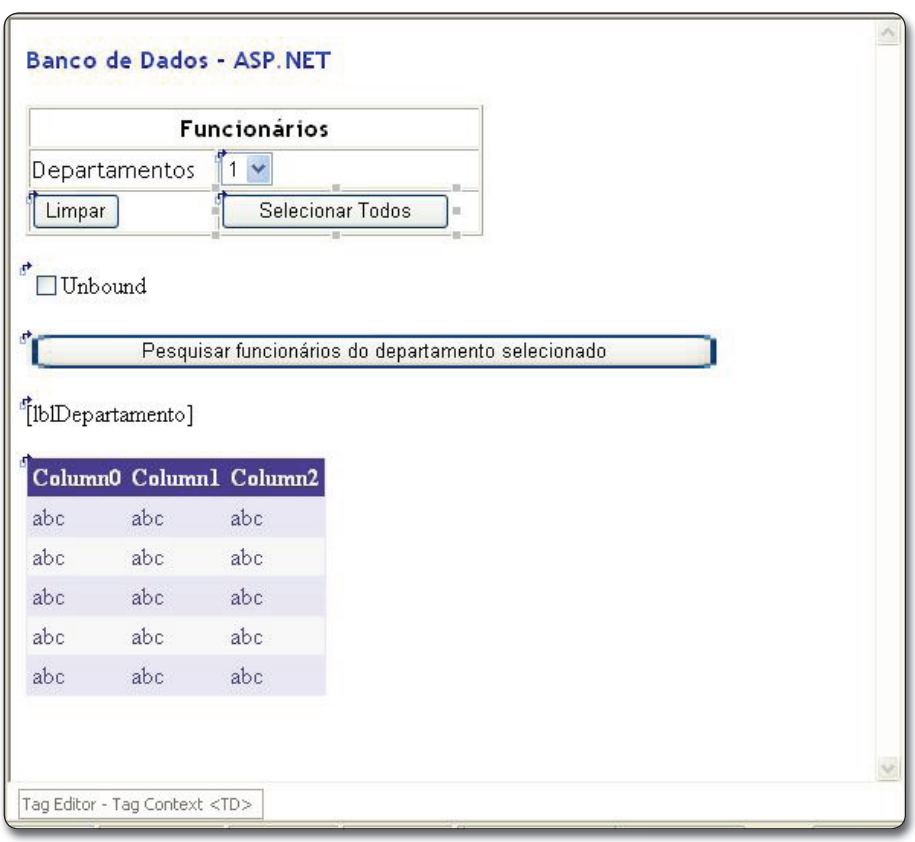


Figura 3. Exemlo de página em nosso exemplo

para *Columns*(colunas), ou seja, estamos criando uma tabela com três linhas e duas colunas.

> **Nota:** Na *Web* as tabelas são normalmente utilizadas para alinhar elementos nas páginas. Elas são representadas pela *tag Table* e possuem outras *tags* tais como: *TD* = célula e *TR* = linha. Cada uma dessas *tags* possui seus próprios atributos/propriedades.

Após inserida a tabela, note que no *Tag Editor* na parte inferior do Delphi (**Figura 5**) são criadas as *tags TR* e *TD* e assim por diante. À medida que vamos adicionando novos elementos à página, este editor é alterado simultaneamente, isso porque ele reflete todas as alterações e converte-as em código HTML, isso significa que se você tem algum conhecimento de HTML ficará bastante fácil de entender.

Perceba as linhas a seguir e veja que foram criadas as *tags TD* e *TR*. Com elas podemos mexer em alinhamentos, cores etc. Nesse momento faremos um pequeno ajuste. Mesclaremos as duas primeiras células para que se tornem uma só.

```
<tr>
   <td><font face="Arial">Funcionários
    </font></td>
   <td><font face="Arial"></font></td>
</tr>
```

Para tanto faça uma pequena alteração: altere a primeira tag TD para

```
<td colspan="2">
```

Por fim exclua a segunda linha *TD* que encontrar. Seu código final ficará como a seguir:

```
   <td colspan="2">
    <font face="Arial">Funcionários</font></td>
```

O que fizemos foi introduzir o comando HTML *colspan* informando a ele quantas células serão usadas para a mesclagem, nesse caso duas. Esse efeito também pode ser notado diretamente na página.

> **Nota:** Caso não esteja visualizando as *tags TD* e *TR* no *Tag Editor*, experimente clicar nas setas para esquerda e/ou direita na barra de ferramentas do *Tag Editor*. Elas mostram todos os elementos HTML do objeto selecionado no Delphi.

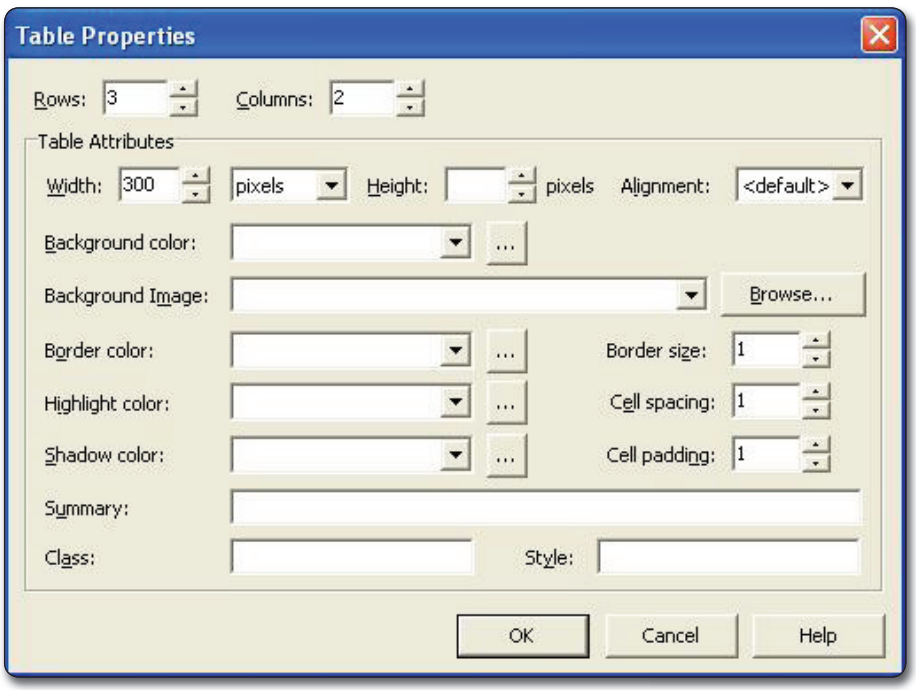


Figura 4. Inclusão de tabela na página

Figura 5. Tag Editor – Editor de tags do Delphi 2006

Logo em seguida digite os textos em seus devidos lugares dentro da tabela de acordo com a **Figura 4**. Para os componentes usaremos a paleta *Web Controls*, portanto, arraste para a célula correspondente um componente *DropDownList* ("DropDepartamentos"). Marque sua propriedade *AutoPostBack* como *True* e adicione a sua propriedade *Items* apenas um elemento onde, tanto no *Text* quanto o *Value*, deverão conter o valor *1*. Faremos uma instrução SQL na tabela *Employee* do *BD* de mesmo nome, que acompanha os exemplos do Firebird, para selecionar os funcionários do departamento escolhido.

Inclua, logo abaixo, um componente do

tipo *CheckBoxList*(“cblDepartamentos”), modifique a propriedade *RepeateColumns* para “3”, assim teremos nossos departamentos mostrados em 3 colunas. Em seguida inclua um *Button* (“btnPesquisar”) fora da tabela e modifique seu *Text* para “Pesquisar funcionários dos departamentos selecionados”. Adicione logo abaixo um *Label*(“lblDepartamento”). Nas duas últimas células da tabela coloque dois *Buttons*(“btnLimpa”) e (“btnSelecionaTodos”), respectivamente. Mude o *Text* de cada *Button* para “Limpar” e “Selecionar todos” também respectivamente. Por último inclua um componente *DataGrid* (“gridEmployee”).

Nota: Para que o *CheckBoxList* fique com uma aparência agradável recomendo que altere as propriedades de fonte e aumente também a largura do componente de forma que ocupe quase toda largura da página.

Nota: Sempre que você quiser mudar de linha pressione ENTER ou SHIFT + ENTER. Existe uma diferença sutil no espaçamento, onde ENTER provoca um novo parágrafo e SHIFT + ENTER provoca uma quebra de linha.

Com relação ao *DataGrid*, você pode formatá-lo de acordo com as opções de formato pré-definidas. Para isso, pressione o botão direito sobre o controle e selecione *Auto Format* ou então a partir do *Object Inspector*. Escolha um formato de acordo com a sua necessidade e observe que as propriedades do estilo serão configuradas automaticamente.

O banco de dados usado como exemplo é o *Employee*, que está instalado no Firebird. O funcionamento se dará da seguinte forma: na primeira vez que a página for carregada, o controle *cblDepartamentos* será preenchido com todos os departamentos existentes na tabela

Listagem 1. Código do evento Page_Load

```
procedure TWebForm2.Page_Load(sender: System.Object;
  e: System.EventArgs);
var
  Conn : FbConnection;
  DataAdapter : FbDataAdapter;
  Ds : DataSet;
  Command : FbCommand;
begin
  { Verifica se foi dado algum Post na página }
  if not Page.IsPostBack Then
  begin

    { Criação dos objetos de conexão }
    Conn := FbConnection.Create;
    DataAdapter := FbDataAdapter.Create;
    Command := FbCommand.Create;

    { Atribuição da string de conexão e abertura do BD}
    Conn.ConnectionString := Conexao;
    Conn.Open;

    { Atribuição dos atributos de seleção dos dados }
    DataAdapter.SelectCommand := Command;
    DataAdapter.SelectCommand.Connection := Conn;
    DataAdapter.SelectCommand.CommandText := 'Select * from Department order by Department';

    { Criaçã em memória do DataSet auxliar }
    Ds := DataSet.Create;
    DataAdapter.Fill(Ds, 'Departamentos');
    try
      { Configura o objet CheckBoxList }
      with cblDepartamentos do
      begin
        { Define que campo será exibido }
        DataTextField := 'Department';
        { Define que campo será armazenado }
        DataValueField := 'Dept_No';
        { Define a origem do controle que é a tabela }
        { Coloca em memória os departamentos }
        DataSource := Ds;
        { Preenche o controle }
        cblDepartamentos.DataBind;
      end;
    finally
      Conn.Close;
    end;
  end;
end;
```

Listagem 2. Código para desmarcar ou selecionar todos os departamentos

```
procedure TWebForm2.ChecarDepartamentos(AChecar: Boolean);
var
  I : Integer;
begin
  for I := 0 to cblDepartamentos.Items.Count-1 do
    cblDepartamentos.Items[I].Selected := AChecar;
  if not AChecar then
  begin
    gridEmployee.Visible := False;
    lblDepartamento.Visible := False;
  end;
end;
```

Listagem 3. Código para desmarcar ou selecionar todas os departamentos

```
procedure TWebForm2.btnSelecionarTodos_Click(sender: System.Object; e: System.EventArgs);
begin
  ChecarDepartamentos(True);
end;
procedure TWebForm2.btnLimpa_Click(sender: System.Object; e: System.EventArgs);
begin
  ChecarDepartamentos(False);
end;
```

Listagem 4. Código do botão de Pesquisa

```
procedure TWebForm2.btnPesquisar_Click(sender: System.Object; e: System.EventArgs);
var
  Conn : FbConnection;
  DataAdapter : FbDataAdapter;
  Ds : DataSet;
  Command : FbCommand;
  DR : FbDataReader;
  Condicao : StringBuilder;
  Selecao : System.Text.StringBuilder;
  I : Integer;
begin
  { Criação dos objetos de conexão }
  Conn := FbConnection.Create;
  DataAdapter := FbDataAdapter.Create;
  Command := FbCommand.Create;
  { Atribuição da string de conexão e abertura do BD}
  Conn.ConnectionString := Conexao;
  Conn.Open;

  { Select para acessar os dados }
  { Atribuição dos atributos de seleção dos dados }
  DataAdapter.SelectCommand := Comand;
  DataAdapter.SelectCommand.Connection := Conn;
  Selecao := System.Text.StringBuilder.Create;
  Selecao.Append('Select Emp_No, First_Name, Last_Name from Employee ');
  Condicao := System.Text.StringBuilder.Create;
  Condicao.Append(' where Dept_no in (');
  for I := 0 to cblDepartamentos.Items.Count-1 do
    if cblDepartamentos.Items[I].Selected then
      { Para usar o QuotedStr no ASP.NET declare o Namespace Borland.Vcl.SysUtils }
      Condicao.Append(QuotedStr( cblDepartamentos.Items[I].Value) + ',');
  Condicao.Append(')');
  Condicao := Condicao.Replace(',)',')');
  Selecao.Append(Condicao);
  Comand.CommandText := Selecao.ToString;
  DR := Comand.ExecuteReader;

  //Exibe o DataGrid de produtos e o Label
  gridEmployee.Visible := True;
  lblDepartamento.Visible := True;

  //Define a origem do DataGrid
  gridEmployee.DataSource := DR;

  //Preenche o DataGrid
  gridEmployee.DataBind;

  //Exibe um texto contendo o número de //produtos do DataGrid
  lblDepartamento.Text := 'Existem ' + gridEmployee.Items.Count.ToString() + ' funcionários.';

  //Fecha o DataReader e a conexão
  DR.Close;
  Conn.Close;
end;
```

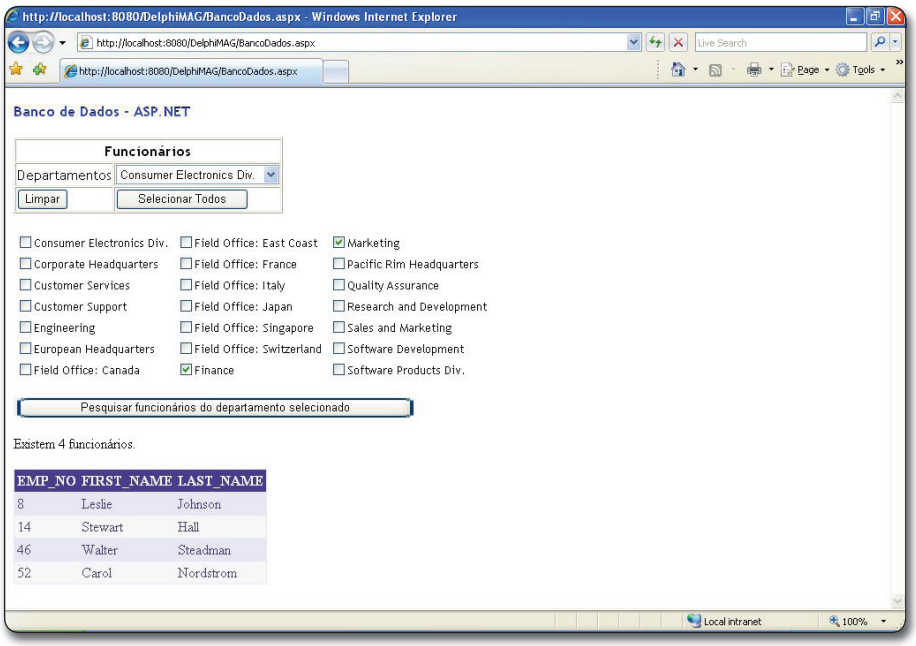


Figura 6. Execução da página

Department do banco de dados. Com isso, o internauta poderá selecionar qualquer departamento existente neste controle.

Quando o botão *btnPesquisar* for clicado serão exibidos no *gridEmployee* todos os funcionários referentes aos departamentos selecionados. Os demais controles existem para que possamos explorar a parte de design interativo com os internautas, ou seja, você pode criar uma página para que o internauta a configure de acordo com as opções selecionadas. Esse processo é possível porque os controles permitem atribuir propriedades em tempo de execução, dispensando completamente a edição manual do design. Esse é um dos pontos fortes do ASP.NET.

Agora que você já conhece o design da página e como deverá funcionar, o próximo passo será digitar os códigos.

Codificando o exemplo

Vamos entrar no editor de códigos do Delphi usando a tecla *F12* ou *View>Show Code*. Como vamos acessar um banco de dados Firebird, declare na seção Uses do formulário a referência da classe do Firebird como segue:

```
FirebirdSql.Data.Firebird
```

Como iremos abrir o banco de dados em dois lugares, para facilitar a declaração da conexão, declare a variável conexão antes do evento *Page_Load*. Essa variável é do tipo *string* e contém o nome do banco de dados (Database), o nome do servidor (Server) e o usuário (User ID) de acordo com suas especificações da instalação do Firebird na sua máquina. Caso necessite, declare a *Password* e os demais parâmetros da conexão. É importante ressaltar que todos os parâmetros declarados na conexão dependem de como você fez a sua instalação do Firebird, os usuários existentes e as senhas. Caso você execute a página e ocorra um erro de *login* ou conexão, verifique estes itens para que o código possa acessar o banco de dados *Employee*.

A constante que criaremos pode ser adicionada logo abaixo a seção *Implementation* como segue:

```
implementation
const
  Conexao = 'User=SYSDBA;
  Password=masterkey;
  Database=C:\Program Files\Firebird\
  Firebird_2_0\examples\empbuild\EMPLOYEE.FDB;';
```

Digite o código da **Listagem 1** no evento *Page_Load* para preencher o controle *cblDepartamentos* com todos os departamentos existentes no banco de dados. Vale dizer que esse evento será executado todas as vezes que a página for carregada. No entanto, os departamentos não mudam, e a pergunta é: por que então carregar todas as vezes que a página for executada? Realmente, não há a menor razão para isso e, desse modo, é possível verificar este fato através do *Page.IsPostBack*, ou seja, tudo o que estiver dentro deste bloco será executado apenas na primeira vez que a página for carregada.

A explicação para o código da **Listagem 1** parece bastante complicado, mas não é. Vamos ver passo a passo o que estamos codificando. Para iniciar estamos criando todos os componentes de conexão em tempo de execução, por isso declaramos as variáveis *Conn*, *DataAdapter, Ds e Command,* como podemos ver a seguir:

```
var
  Conn : FbConnection;
  DataAdapter : FbDataAdapter;
  Ds : DataSet;
  Command : FbCommand;
```

Em seguida atribuímos a *string* de conexão ao componente *FbConnection*, conectamos ao *BD* e atualizamos os demais componentes de conexão com os atributos necessários para seleção dos dados. Note que criamos um *DataAdapter* e *FbCommand*. Logo após atribuímos a propriedade *Select Command* do *DataAdapter* do *FbCommand* recém criado. O *DataAdpater* é o componente que usamos para selecionar (*SelectCommand*), atualizar (*UpdateCommand*), inserir (*InsertCommand*) e apagar(*DeleteCommand*) os registros na tabela do banco de dados. Como pode ver podemos ter as quatro operações básicas de banco de dados em um único componente, mas para que cada operação seja executada precisamos ter um *FbCommand*, por isso o criamos e o atribuímos à propriedade *SelectCommand*.

Depois da conexão estabelecida, criamos um *DataSet* em memória, o preenchemos e por último atualizamos o controle *CheckBoxList*. Mas o que é um *DataSet*? *DataSet* é uma representação de uma ou mais tabelas na memória (contendo os métodos *Select, Insert, Delete* e *Update*) e é possível inclusive estabelecer relacionamentos entre elas. Isso significa dizer que podemos ter um *DataSet* contendo diversas tabelas.

Listagem 5. Trecho de código de preenchimento do DropDepartamentos no evento Page_Load

```
procedure TWebForm2.Page_Load(sender: System.Object; e: System.EventArgs);
begin
...
   with DropDepartamento do
   begin
     Items.Clear;
     DataTextField := 'Department';
     DataValueField := 'Dept_No';
     DataSource := Ds;
     DropDepartamento.DataBind;
   end;
...
```

Pelo fato dessa representação ser feita na memória, tudo é mais rápido. A grande característica do ADO.NET é trabalhar com dados desconectados, ou seja, todas as alterações são feitas na memória até que você as efetive no banco de dados.

Voltando ao exemplo: como todos os departamentos estão listados e dispõem de um *CheckBox* para cada departamento, vamos facilitar a vida do internauta. Caso o internauta queira selecionar todos os departamentos existentes, para que ele não precise clicar em cada departamento, podemos montar um código que faça isso automaticamente. A mesma idéia serve para desmarcar todos os departamentos selecionados.

Na **Listagem 2** faremos a criação de um procedimento para selecionar ou limpar a seleção de todos os *CheckBoxes* do *cblDepartamentos*. Note que estamos recebendo um parâmetro (*AChecar*) do tipo *Boolean*, dessa forma podemos fazer a chamada ao procedimento tanto do botão *btnLimpa*(*False*) quanto para *btnSelecionarTodos*(*True*). Localize então a seção *Public* no Delphi e declare o procedimento como a seguir:

```
procedure ChecarDepartamentos(AChecar: Boolean);
```

Para criar o cabeçalho da implementação do procedimento pressione *Ctrl + Shift + C*. Digite o código da **Listagem 2**. A providência que tomamos na **Listagem 2** é bem simples. Apenas envolvemos em um laço *For* os itens do componente *cblDepartamentos* e atribuímos o valor do parâmetro *AChecar* marcando ou desmarcando os itens de acordo com o valor atribuído ao parâmetro. Em seguida ocultamos o *lblFuncionarios* e o *gridFuncionarios* de acordo com o valor do parâmetro.

Agora basta chamarmos a prodecure *ChecarDepartamentos* passando para ela *True* ou *False* dependendo da situação. Para o botão *btnLimpa* digite a chamada do procedimento passando *False* no parâmetro e para *btnSelecionarTodos* repita a chamada passando *True*. Veja na **Listagem 3** o código completo de ambos botões.

A grande facilidade do ASP.NET é demonstrada aqui. Após selecionar os departamentos, você precisa montar o *DataGrid* com todos os funcionários, de acordo com os respectivos departamentos selecionados. A construção desse código é relativamente simples porque basta você aplicar um filtro à cláusula *Where* do SQL. No entanto, como o internauta pode selecionar uma ou todos os departamentos, o filtro precisará ser montado de forma que o *looping* percorra todos os itens do *cblDepartamentos* e verifique quais estão selecionados. Para cada item selecionado, é concatenada uma variável que contém o respectivo *ID* do departamento para executar o *DataReader* com a string *SQL*.

Quando usar um *DataReader*? Use o *DataReader* sempre que precisar ler uma fonte de dados; o *DataReader* é do tipo *Forward Only*, ou seja, você não consegue navegar em registros anteriores. Em termos de performance, um *DataReader* é muito mais rápido que um *DataAdapter*. A desvantagem é que você não conseguirá aplicar paginação em um *DataGrid* se este tiver sido preenchido com um *DataReader*, como é o caso deste exemplo.

Você irá notar que, na concatenação de *strings*, estou usando a classe *System.Text.StringBuilder*. Sempre que você precisar concatenar *strings*, opte pelo *StringBuilder* ele é infinitamente mais rápido que outras formas de concatenação, por exemplo, variável := variável + conteúdo. Na **Listagem 4** podemos conferir o código completo do botão de pesquisa.

Vamos entender a **Listagem 4**. Note que

no início o código se parece muito com o código do evento *Page_Load*. E realmente é bem parecido. No início declaramos as variáveis necessárias para conexão e seleção dos dados do BD. As novidades são que declaramos 3 novas variáveis:

```
var
  DR : FbDataReader;
  Condicao : StringBuilder;
  Selecao : System.Text.StringBuilder;
```

Declaramos um *FbDataReader*. O *FbDataReader* executa a pesquisa no banco e traz os dados em memória. Com seus dados carregados atualizamos nosso *DataGrid*. Duas variáveis do tipo *StringBuilder* que receberão as *strings* de filtro, ou seja, a instrução *SQL* propriamente dita. Criamos então os componentes em memória, atualizamos as devidas conexões e filtros. Filtrados os dados, atribuímos o *FbDataReader* ao *DataGrid* e mostramos os dados em tela.

Após digitar todo o código, clique com o botão direito em *BancoDados.aspx* e selecione *Set As Start Page*. Compile o projeto e pressione *F9* para que o Delphi execute o exemplo e abra o *browser*.

A última alteração que fiz foi incluir os departamentos da tabela *Department* também no *DropDepartamentos*. O procedimento é bem simples: Basta repetir o *with..do* que fizemos no evento *Page_Load* mudando somente as referências, ou seja, mudando de *cblDepartamentos* para *DropDepartamento*. Veja na **Listagem 5** como fica a instrução completa. Insira-a logo após o bloco *with..do* do *clbDepartamentos*. O código completo do evento *Page_Load* depois da alteração você encontra na **Listagem 6**. Na **Figura 6** podemos ver a página finalizada e em execução.

Conclusão

O ASP.NET nos permite interagir com o internauta de forma simples e objetiva com apenas algumas linhas de programação, personalizando a exibição do conteúdo da página para o internauta. Quanto ao acesso a fontes de dados, use e abuse das classes e controles de que o ASP.NET dispõe; afinal, programar para *Web* está cada vez mais simples, fácil e produtivo. ●

Listagem 6. Evento completo Page_Load

```
procedure TWebForm2.Page_Load(sender: System.Object; e: System.EventArgs);
var
  Conn : FbConnection;
  DataAdapter : FbDataAdapter;
  Ds : DataSet;
  Command : FbCommand;
begin
  { Verifica se foi dado algum Post na página }
  if not Page.IsPostBack Then
  begin
    { Criação dos objetos de conexão }
    Conn := FbConnection.Create;
    DataAdapter := FbDataAdapter.Create;
    Command := FbCommand.Create;

    { Atribuição da string de conexão e abertura do BD}
    Conn.ConnectionString := Conexao;
    Conn.Open;

    { Select para acessar os dados }
    { Atribuição dos atributos de seleção dos dados }
    DataAdapter.SelectCommand := Command;
    DataAdapter.SelectCommand.Connection := Conn;
    DataAdapter.SelectCommand.CommandText := 'Select * from Department order by Department';

    { Criaçã em memória do DataSet auxliar }
    Ds := DataSet.Create;
    DataAdapter.Fill(Ds, 'Departamentos');
    try

      { Configura o objet CheckBoxList }
      with cblDepartamentos do
      begin
        { Define que campo será exibido }
        DataTextField := 'Department';
        { Define que campo será armazenado }
        DataValueField := 'Dept_No';
        { Define a origem do controle que é a tabela }
        { Coloca em memória os departamentos }
        DataSource := Ds;
        { Preenche o controle }
        cblDepartamentos.DataBind;
      end;

      with DropDepartamento do
      begin
        Items.Clear;
        DataTextField := 'Department';
        DataValueField := 'Dept_No';
        DataSource := Ds;
        DropDepartamento.DataBind;
      end;
    finally
      Conn.Close;
    end;
  end;
end;
```

Nesta seção você encontra artigos sobre a linguagem PHP e a ferramenta Delphi for PHP

Delphi for PHP

Crie um site de publicação de notícias – Parte 4

Fabrício Desbessel
(fabricio@fabricio.pro.br)
é professor de Linguagem de Programação do Curso Técnico em Informática do Colégio Frederico Jorge Logemann de Horizontina/RS e da FAHOR Faculdade Horizontina. Delphiano de coração está sempre disposto a provar que com o Delphi sempre teremos a melhor solução. Site www.fabricio.pro.br.

Na nossa quarta parte do mini-curso vamos criar a parte principal do nosso sistema, a área de acesso ao visitante. Nessa parte é muito importante o visual e a usabilidade, o que, na maioria das vezes nós programadores deixamos a desejar. Na verdade o *Design* de um site deve ser uma criação artística, desenvolvida por profissionais habilitados para essa área. O que irei demonstrar é como separar a programação e o desenvolvimento do visual de forma bem transparente e prática para uma equipe com profissionais das duas áreas, e para isso utilizaremos *templates*.

Idéia de Layout

Antes de começar a programar o site é necessário ter uma idéia de como será sua aparência. Para tanto, sugiro que seja desenhado um esboço das partes, o que pode ser chamado de *Wireframe*. De forma simples trace idéias de como poderá ser o site. Quanto mais idéias melhor. Depois, com toda sua equipe estude-as e defina o que pode ser aproveitado de cada uma. Para nosso mini-curso utilizei o Excel para criar o *Wireframe* exibido na **Figura 1**.

Programando

Primeiramente vamos fazer o que mais gostamos, a programação. Abra o projeto realizado até agora e adicione um novo formulário através do menu *File|New>Form*. Altere o *Name* desse formulário para "UIndex" e salve-o "index.php". Acesse o menu *File>Use Unit* e escolha *dados.php*.

Para fazer a lista das últimas dez notícias adicione ao formulário um componente *Query*, da paleta de componentes *Data Access*. Altere a propriedade *Name* para "sqlUltimas", ligue a propriedade *Database* a *UDados.dbjornal1* e digite a seguinte instrução *SQL*:

```
Select titulo, data_publicacao, id_noticia
From noticias
```

Ainda com o componente *sqlUltimas* selecionado defina a propriedade *Order Field* para "data_publicacao" e a propriedade *Order* para "desc". Com essas definições o Delphi for PHP adicionará a instrução "Order By data_publicacao desc" à nossa sentença *SQL*. Verifique a propriedade *LimitCount* que por padrão deve estar "10". Essa propriedade define a quantidade de registros que serão pesquisados no banco de dados. Por último, defina a propriedade *Active* desse componente como *True*.

Adicione um componente *Data Source* e altere a propriedade *Name* para "dsUltimas" e a propriedade *DataSet* ligue à *sqlUltimas*. Insira no formulário um componente *DBRepeater*, encontrado na paleta de componentes *Data Controls*. Altere seu *Name* para "dbrUltimas" e ligue sua propriedade *DataSource* para *dsUltimas*. Também apague sua propriedade *Caption*.

Dentro do *DBRepeater* adicione dois *Label's*, um abaixo do outro. Para o segundo, defina a propriedade *DataSource* para *dsUltimas* e a propriedade *DataField* como "titulo". A idéia do primeiro *Label* é mostrar a data formatada para o Brasil e, por isso não podemos fazer a ligação direta ao campo. Então, no evento *OnBeforeShow* desse *Label* coloque o código a seguir.

```
global $UDados;
$Data=explode("-", $this->sqlUltimas->
  data_publicacao);
$Data=array_reverse($Data);
$this->Label1->Caption=implode("/", $Data);
```

No código anterior utilizou-se a função *explode* para separar os valores e retirar o traço que vem na data pesquisada no banco de dados MySQL. A função *array_reverse* inverte a ordem das *strings* deixando na posição, normalmente utilizada no Brasil. E a função *implode* coloca as barras entre as partes juntando-as.

Selecione o *Label* ligado ao título e, no evento *OnBeforeShow* coloque o código a seguir que fará o link para a própria página passando o *id* da notícia clicada como campo de consulta.

```
$this->Label2->
  Link="index.php?id_noticia=".
$this->sqlUltimas->id_noticia;
```

Altere a propriedade *Active* do *sqlUltimas* para *True* e compile para testar o funcionamento. Claro que para isso é interessante ter cadastrado mais de 10 notícias.

Agora adicione ao formulário outro componente *Query* e altere as propriedade *Name* para "sqlNoticia" e *DataBase* para "UDados.dbjornal1". Na propriedade *SQL* informe a sentença apresentada a seguir:

```
SELECT n.id_noticia, n.titulo, n.texto,
  u.nome, n.data_publicacao, n.data_alteracao,
  n.fonte, n.qtde_visualizacoes,
  n.qtde_impressoes, n.qtde_envio
FROM noticias n, usuarios u
WHERE n.id_usuario=u.id_usuario
```

Lembrando que sempre é recomendável selecionar somente os campos que serão realmente exibidos ao usuário. Via código iremos fazer um filtro para exibir, inicialmente a última notícia cadastrada e, quando o usuário clicar na lista, mostrar a notícia selecionada.

Coloque um *DataSource* ao formulário e ligue a propriedade *DataSet* para *sqlNoticia*, alterando também seu *Name* para *dsNoticia*. Para colocar os campos que irão exibir os dados da notícia vamos utilizar, como base um *DBRepeater*. Isso parece estranho, pois será uma única notícia e não várias, mas nesse caso o *DBRepeater* irá funcionar como um *container* que poderá ser facilmente movido de posição ou redimensionado.

Altere o *Name* do *DBRepeater* para *dbrNoticia* e ligue sua propriedade *DataSource* para *dsNoticia*. Esse *DBRepeater* pode ser colocado abaixo do já existente não importando com nosso *layout* pois essa disposição será ajustada com a utilização do *Template*. Apenas atente para o tamanho dos *DBRepeater*, sendo que na página estarão lado a lado.

Agora dentro do *DBRepeater* temos que colocar os *Labels* para exibir as informações. Basta colocar os mesmos e configurar as propriedades *DataSource* e *DataField*. Para os *Labels* que irão exibir as datas (publicação e alteração) simplesmente defina o *Name*, para depois, via código passar a data no nosso formato. Sugiro os nomes começando como lbl+campo (ex: *lblData_Publicacao*). Faça o mesmo para o campo *Nome* que trará o autor da notícia, possibilitando a colocação do texto "Publicado por", antes do nome.

Para o campo texto da notícia utilize um componente *Memo*, alterando a propriedade *ReadOnly* para *True*, e *BorderStyle* para *bsNome*. Utilize para orientação a **Figura 2**.

No campo do título, altere a propriedade *Font.Weight* para *Bold*. Nos campos de data coloque *fsItalic* na propriedade *Font.Style*. Nos campos *título, texto* e *fonte* aumente a largura deixando-os com um tamanho muito próximo do *DbRepeater*.

No final adicione um componente *Image* e altere sua propriedade *AutoSize* para *True*. Também coloque no *Hint* o texto "Clique aqui para ver todas as fotos da notícia" e mude a propriedade *ShowHint* para *True*. Deixe um bom espaço para que a imagem possa ser exibida na sua totalidade. No evento *OnBeforeShow* do formulário coloque o código contido na **Listagem 1**.

O código da **Listagem 1** verifica se foi informado um campo de consulta no *Link*

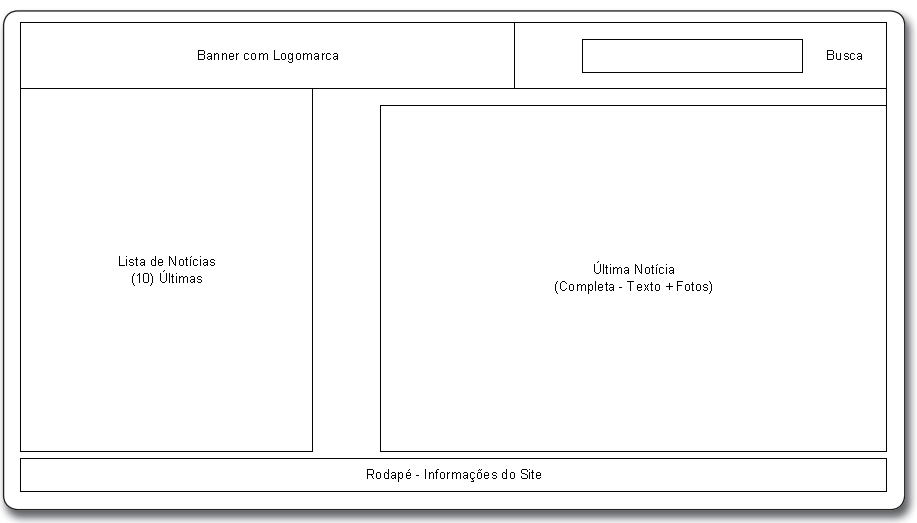


Figura 1. Wireframe da parte visível

Listagem 1. Código do evento OnBeforeShow do formulário

```
$this->sqlNoticia->Close();
$codigo=$this->input->id_noticia;
if (is_object($codigo)) {
   $this->sqlNoticia->Filter="id_noticia=".$codigo->asInteger();
}else{
   $this->sqlNoticia->Filter="id_noticia=(Select Max(id_noticia) From noticias)";
}
$this->sqlNoticia->Open();
```

Listagem 2. Código do evento OnBeforeShow do campo titulo

```
global $UDados;
$Atual=$this->sqlNoticia->qtde_visualizacoes;
$Atual=$Atual+1;
$UDados->dbjornal1->execute("Update noticias Set qtde_visualizacoes=".
$Atual." Where id_noticia=".
$this->sqlNoticia->id_noticia);
```

Listagem 3. Código do evento OnBeforeShow do Image

```
$diretorio = "C:/CD/Jornal/fotos/n".$this->sqlNoticia->id_noticia."/";
if(is_dir($diretorio)){
  if (file_exists($diretorio."inicial.jpg")){
    $this->Image1->ImageSource="fotos/n".
    $this->sqlNoticia->id_noticia."/inicial.jpg";
    $this->Image1->Link="fotos.php?id_noticia=".
    $this->sqlNoticia->id_noticia;
  }else{
    $this->Image1->ImageSource="fotos/semfoto.jpg";
    $this->Image1->Link="";
  }
}
```

Listagem 4. Código do evento OnBeforeShow do campo data_publicacao

```
setlocale(LC_ALL,"pt_BR", "ptb");
$Data=strtotime($this->sqlNoticia->data_publicacao);
$this->lblData_Publicacao->Caption=
strftime("Publicada em %d de %B de %Y", $Data);
```

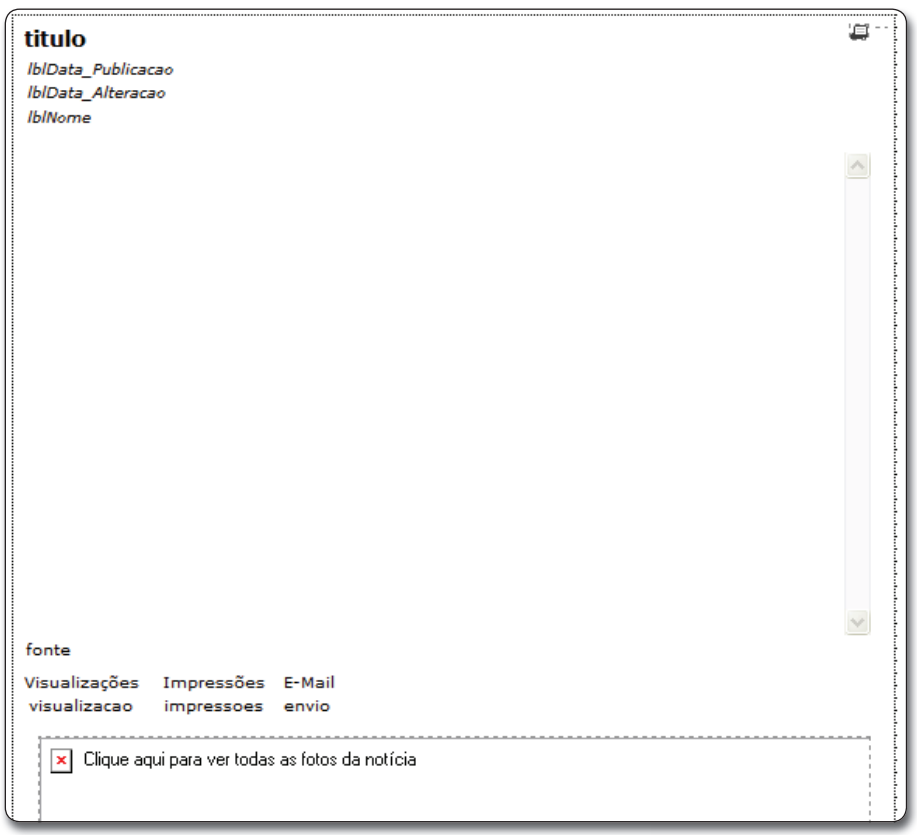


Figura 2. Parte da notícia

o que resultará no filtro pelo *id* da notícia. Se não existir o campo, será selecionada a última notícia cadastrada, e por isso, tem-se uma *sub-consulta* no *Filter*.

A cada vez que uma notícia é exibida precisamos incrementar o campo *qtde_visualizacoes*. Para fazer isso acesse o evento *OnBeforeShow* do campo de título da notícia e codifique conforme é disposto na **Listagem 2**.

Explicando o código da **Listagem 2**, tem-se a possibilidade de executar uma instrução diretamente no banco de dados através do método *Execute* do componente *DataBase*. Então elaboramos uma instrução *SQL* de *Update*, onde faz-se uma soma da quantidade atual mais um. Com isso temos nosso contador de visualização.

Para exibição de uma foto da notícia, definiu-se que será exibida uma foto com o nome *inicial.jpg* que deve ter sido enviado no momento de cadastro de notícias. Lembrando que no cadastro de notícia pode-se enviar várias fotos, mas na página inicial do site será exibida somente a imagem com o nome *inicial.jpg*, que fará um *link* para a visualização de todas as fotos. Então para mostrar essa foto codifique a **Listagem 3** no evento *OnBeforeShow* do componente *Image*.

No código da **Listagem 3** define-se o diretório onde deve estar a imagem, sendo que, dentro da raiz do site deve existir um diretório *fotos* e dentro um diretório que, seu nome inicie com a letra "n" seguida de seu código(*id_noticia*). É essencial testar se existe o diretório (*is_dir*) pois pode ser que a notícia não possua fotos e também verificar se existe uma foto com o nome *inicial.jpg* (*file_exists*).

Depois é só informar a propriedade *ImageSource* e *Link*, onde criou-se uma chamada para uma página de nome *fotos.php*. Se não existir a foto inicial será exibida uma imagem avisando. Você também pode optar por uma imagem transparente.

Para os campos de datas também utiliza-se o evento *OnBeforeShow* para formatar a data e exibi-la. Na **Listagem 4** e **5** tem-se os códigos para cada campo de exibição das datas.

Nos códigos das **Listagens 6 e 7** utilizou-se a função *setlocale* que serve para definir o local (país) padrão. O *pt_BR* e *ptb* são valores para o Brasil, sendo que o primei-

ro serve para servidores com Linux e o segundo para servidores com Windows. A função *strtotime* transforma o texto em um *timestamp* e *strftime* formata a data levando em conta as configurações regionais, permitindo a inclusão de textos. Para *data_alteracao* faz-se um teste para verificar se o campo está preenchido (*if strlen*), através da verificação do seu tamanho.

Agora, adicione dois componente *Image* no *dbrNoticia*, no canto superior direito, para funcionar como botões de impressão e envio por e-mail. Na propriedade *ImageSource* selecione uma imagem que represente uma impressora para o primeiro e um e-mail para o segundo.

Selecione o *Image* referente à impressão e na propriedade *Hint* coloque "Versão para impressão", *ShowHint* como *True*. No seu evento *OnBeforeShow* digite o código a seguir:

```
$this->Image2->
  Link="impressao.php?id_noticia=".
$this->sqlNoticia->id_noticia;
$this->Image2->LinkTarget="_blank";
```

Repita os passos anteriores para o outro *Image* que refere-se ao envio por e-mail, definindo o *Hint* como "Envie essa notícia". Acesse o evento *OnBeforeShow* e codifique conforme o que segue:

```
$this->Image3->Link="envio.php?id_noticia=".
$this->sqlNoticia->id_noticia;
$this->Image3->LinkTarget="_blank";
```

Nos códigos descritos anteriormente simplesmente definiu-se o *Link* para as páginas que deverão ser criadas com o intuito de imprimir e enviar a notícia, passando-se o *id* da notícia como campo de consulta.

Listagem 5. Código do evento OnBeforeShow do campo data_alteracao

```
if(strlen($this->sqlNoticia->data_alteracao) > 0){
   setlocale(LC_ALL,"pt_BR", "ptb");
   $Data=strtotime($this->sqlNoticia->data_alteracao);
   $this->lblData_Alteracao->Caption=
   strftime("Alterada em %d de %B de %Y", $Data);
}else{
  $this->lblData_Alteracao->Caption="";
}
```

Figura 3. Template para aplicação

Figura 4. Página Index em execução

Listagem 6. Código HTML da página modelo (template)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
  Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2" />
<title>Sistema de Notícias</title>
{$HeaderCode}
</head>

<body>
{$StartForm}
  <table width="770" border="0" align="center" cellpadding="0" cellspacing="0">
    <tr>
      <td><table width="100%" border="0" cellspacing="0" cellpadding="0">
        <tr>
          <td><center>

            <table width="100%" border="0"
              cellspacing="0" cellpadding="0">
              <tr>
                <td width="54%"><div align="left">
                  <img src="clubedelphi.gif" alt="" width="400" height="72"/></div></td>
                <td width="46%"><div align="right">
                {$edtBusca} {$btnBusca}</div></td>
              </tr>
            </table>

          </center>
          </td>
        </tr>
        <tr>
          <td>

          <table width="100%" border="0" cellspacing="0" cellpadding="0">
            <tr>
              <td width="39%" valign="top">
                <div align="left">{$dbrUltimas}
                </div></td>
              <td width="61%" valign="top">
                <div align="left">{$dbrNoticia}
                </div></td>
            </tr>
          </table>

            <table width="100%" border="0"
              cellspacing="0" cellpadding="0">
              <tr>
                <td bgcolor="#F7C500"><center>
                  <b><font size="2" face="Verdana, Arial, Helvetica, sans-serif">
                    Sistema de Not&iacute;cias - Clube Delphi - DevMedia</font></b>
                </center>
                </td>
              </tr>
            </table></td>

        </tr>
      </table></td>
    </tr>
  </table>
{$EndForm}
</body>
</html>
```

Listagem 7. Código do evento OnBeforeShow do Label da página fotos.php

```
  $codigo=$this->input->id_noticia;
  if (is_object($codigo)) {
    $diretorio = "C:/CD/Jornal/fotos/n".$codigo->asInteger()."/";

    if (is_dir($diretorio)) {

      if ($dh = opendir($diretorio)){
         while (($file = readdir($dh))!== false){

           if (filetype($diretorio.$file)== "file"){

              if (getimagesize($diretorio.$file)){
                $Lista.="<img src='fotos/n".$codigo->asInteger()."/".$file."'><hr>";
              }
           }
         }
      }
    }
    $this->Label1->Caption=$Lista;
  }
```

Para fazer a busca adicione um componente *Edit* e um *Button*. Mude o nome do *Edit* para "edtBusca". Como *Name* do *Button* defina "btnBusca". Clique duas vezes sobre o botão e codifique-o conforme podemos ver em seguida:

```
$this->sqlUltimas->Filter=
  'Upper(titulo) like Upper("'.
$this->edtBusca->Text.'%")';
$this->sqlUltimas->Refresh();
```

Para realizar a busca será aproveitado o componente de nome *sqlUltimas* mostrando os registros encontrados no lugar das dez últimas notícias.

Criando o template

Essa funcionalidade é fenomenal, pois possibilita que os trabalhos de *Design* e programação sejam realizados por pessoas diferentes, de forma muito transparente e ágil. O Delphi for PHP utiliza um *framework* chamado *SmartyTemplate* que utiliza um HTML como base para montar a página PHP. Assim o desenho fica no HTML e a programação no PHP.

Para utilizar essa funcionalidade é necessário, primeiramente criar uma página. Abra seu software preferido para edição HTML e crie uma página levando em conta o modelo da **Figura 1**.

Para organização das partes utilize as *tags Table* ou *Div*, ficando, a seu critério. Depois de montar precisamos adicionar as *Tags* para os componentes que iremos posicionar nessa página. Para definir uma *Tag* iniciamos com chaves "{" seguido de cifrão mais o nome do componente. Exemplo: *{$btnBusca}*.

Existem algumas *Tags* obrigatórias que devem ser colocadas nas páginas: *{$HeaderCode}*, *{$StartForm}* e *{$EndForm}*. A *HeaderCode* deve ser colocada dentro da *Tag <HEAD>*. A *StartForm* abaixo da *Tag <BODY>* e *EndForm* antes da *Tag </BODY>*.

Na **Figura 3** tem-se a página sendo visualizada no editor HTML e na **Listagem 6** tem-se o código HTML da mesma.

Note na **Figura 3** o nome dos componentes que iremos utilizar. O interessante é que os *DBRepeater's* englobam os componentes que estão dentro, diminuindo a necessidade de adição de *Tags*. Por isso utilizou-se um *DBRepeater* para exibir a notícia, mesmo tendo a certeza que será sempre um único registro.

Com a página pronta salve o arquivo HTML, informando o nome como "index.html", no mesmo diretório da aplicação. Agora volte para o Delphi for PHP, acesse a página *index.php*, clique sobre ela para selecionar e altere a propriedade *TemplateEngine* para *SmartyTemplate*. Logo abaixo, na propriedade *TemplateFilename* coloque *index.html*. Pronto, com isso finalizamos essa página. Execute-a e veja como ficou (**Figura 4**).

Criando a página de fotos

Crie um novo formulário através do menu *File|New>Form*. Altere seu *Name* para "UFotos" e salve como "fotos.php". Adicione um *Label* no formulário e altere sua propriedade *Width* para "785" e *Alignment* para *agCenter*. No seu evento *OnBeforeShow* coloque o código contido na **Listagem 7**.

No código da **Listagem 7** criou-se uma variável *Lista* que é populada a cada iteração com uma *Tag IMG*, que nada mais é que uma imagem no HTML. Ao final passamos o conteúdo dessa variável para o *Caption* do *Label*. Isso é muito interessante, pois possibilita a escrita de códigos HTML dentro dos componentes aumentando as possibilidades. Pode-se dizer que não temos limitações, pois com isso podemos fazer qualquer coisa no Delphi for PHP.

Criando a página de impressão

Para a página de impressão vamos reutilizar todo o *DBRepeater* de nome *dbrNoticia* da página *index.php*. Então crie um novo formulário, mude seu *Name* para "UImpressao" e salve o arquivo como "impressao.php". Copie da página *index.php* os componentes *dbrNoticia*, *sqlNoticia* e *dsNoticia* para colar nessa página. No evento *OnBeforeShow* coloque o código da **Listagem 8**.

O Código da **Listagem 8** é uma cópia do evento *OnBeforeShow* da página *index.php*, acrescentado o evento *OnBeforeShow* do campo título, responsável pela execução da instrução *SQL* que irá incrementar o campos *qtde_impressoes*. Como esse código faz referência a *Unit UDados* é necessário acessar o menu *File>Use Unit* e escolher *dados.php*. Para a exibição correta da imagem é necessário copiar o código do evento *OnBeforeShow* do componente

Listagem 8. Código do evento OnBeforeShow da página de impressão

```
$this->sqlNoticia->Close();
$codigo=$this->input->id_noticia;

  if (is_object($codigo)) {
     $this->sqlNoticia->Filter="id_noticia=".$codigo->asInteger();
  }else{
     $this->sqlNoticia->Filter="id_noticia=(Select Max(id_noticia) From noticias)";
  }

$this->sqlNoticia->Open();
global $UDados;
$Atual=$this->sqlNoticia->qtde_impressoes;
$Atual=$Atual+1;
$UDados->dbjornal1->execute("Update noticias Set qtde_impressoes=
  ".$Atual." Where id_noticia=". $this->sqlNoticia->id_noticia);
```

Listagem 9. Código do evento OnBeforeShow da página de envio do e-mail

```
$this->sqlNoticia->Close();
$codigo=$this->input->id_noticia;

if (is_object($codigo)) {
   $this->sqlNoticia->Filter="id_noticia=".$codigo->asInteger();
   $this->sqlNoticia->Open();
}
```

Listagem 10. OnClick do botão Enviar

```
$server="http://localhost/";
$to_email=$this->edtSeuEmail->Text;
$from=$this->edtSeuEmail->Text;
$from_email=$this->edtSeuEmail->Text;
$comment=$this->memComentario->Text;
$link = $server. "index.php?id_noticia=".$this->sqlNoticia->id_noticia;
$message="$from achou que a noticia abaixo seria interessante para você.\n\n".
$this->sqlNoticia->titulo."\n\nPara acessá-la clique no link abaixo:\n".$link;

if($comment != "") {
   $message.="\n\nComentário:\n$comment";
}

$subject="Notícia interessante";
$add="From: ".$from_email." \n"."Reply-To: ".$from_email."\n";

if (mail ($to_email, $subject, $message, $add)) {
   $msg =  "Notícia enviada com sucesso!<br><br>Obrigado!";
   global $UDados;
   $Atual=$this->sqlNoticia->qtde_envio;
   $Atual=$Atual+1;
   $UDados->dbjornal1->execute("Update noticias Set
     qtde_envio=".$Atual." Where id_noticia=".$this->sqlNoticia->id_noticia);
}else{
  $msg = "Erro Interno. Tente novamente mais tarde.";
}

$this->lblMensagem->Caption=$msg;
redirect('envio.php?id_noticia=?'.$this->sqlNoticia->id_noticia);
```

Image1 do formulário *index.php*.

Tome cuidado com o nome da imagem no novo formulário pois se ela não for mais *Image1* você terá que alterar o código e também retire a linha que adiciona um *Link*. Também copie os códigos dos campos *data_publicacao*, *data_alteracao* e *nome*. Nessa página de impressão opte por trocar o *Memo* por um outro *Label* porque precisamos do texto todo descrito para a impressão correta, o que não acontece com o *Memo*.

Clique na página e no *Object Inspector* entre na aba *Javascript* e acesse o evento *OnLoad*. Abaixo da linha *//Add your javascript code here* coloque o código a seguir que chamará a impressão e depois fechará a janela:

```
print();
close();
```

Enviando e-mail

Para o envio crie um novo formulário, altere sua propriedade *Name* para "UEnvio" e salve o arquivo como "envio.php". Copie os componentes *sqlNoticia* e *dsNoticia* da página *index.php* para essa página e altere a propriedade *SQL* para retirar os campos e ligações que não precisaremos mais. Ao final, a instrução deve ser a seguinte:

```
SELECT id_noticia, titulo, qtde_envio
FROM noticias
```

Lembre que deve ser acessado o menu *File>Use Unit* para ligar essa página à *dados.php*. Adicione um *Label* configurando sua propriedades *DataSource* e *DataField* para ligar ao campo texto. Agora adicione um *Label* e um *Edit* para informar seu nome, seu e-mail, nome do destinatário e e-mail do destinatário. Para o *Name* dos *Edit's* defina *edtSeuNome*, *edtSeuEmail*, *edtDestinatario* e *edtEmailDestinatario*, respectivamente. Adicione também um *Memo* e defina seu *Name* como "memComentario". Abaixo, adicione um *Button* e altere o *Caption* para enviar. Veja na **Figura 5** como deve ficar essa página. No evento *OnBeforeShow* do formulário coloque o código da **Listagem 9** que abrirá a notícia.

Agora vamos ao item final, codificar o envio do e-mail. Para tanto, ainda é necessário colocar um novo *Label* no formulário, alterando seu *Name* para "lblMensagem" e deixando seu *Caption* vazio. Faça a codificação do evento *OnClick* do botão conforme a **Listagem 10**.

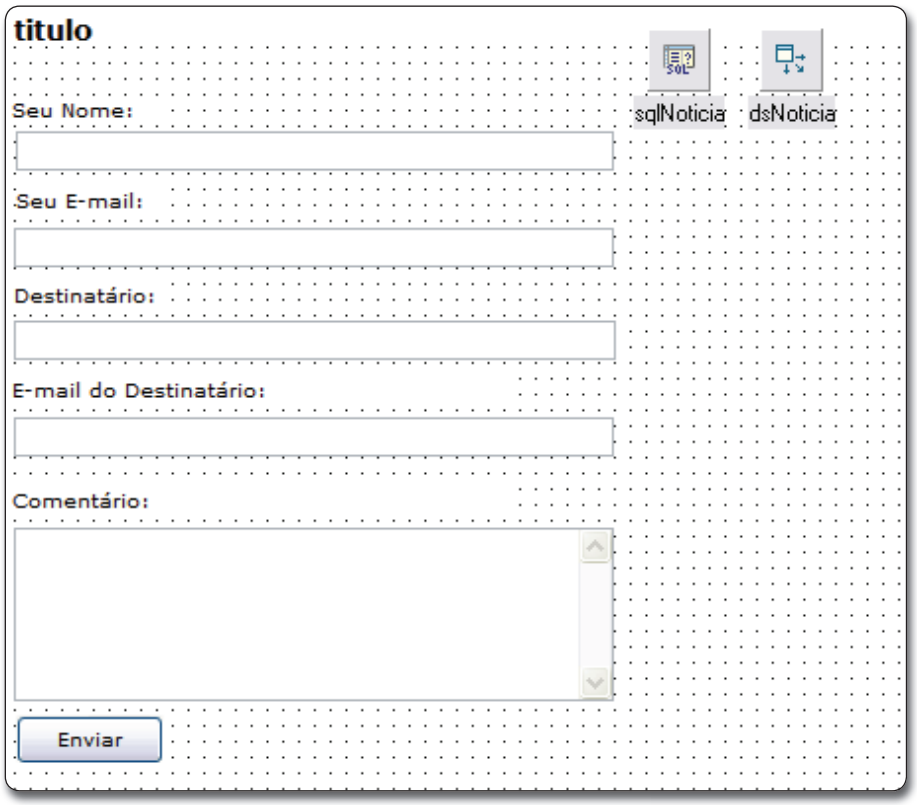


Figura 5. Formulário para envio de e-mail

No código da **Listagem 10** tem-se, inicialmente a declaração de variáveis (*$from*, *$to_email*, *$subject* etc.) necessárias para montar o e-mail de forma completa, com cabeçalho e texto. Como texto do e-mail tem-se a mensagem que alguém achou que a notícia seria interessante para você, juntamente com o link da notícia para o acesso direto. Com as variáveis preenchidas, testa-se o comando mail que é o responsável por enviar o e-mail, onde se passa a informação do destinatário, assunto, texto da mensagem e emitente. Se o envio acontecer executa-se um update na tabela de notícias para incrementar o contador de envios. Ao final da execução do código tem-se como resposta uma mensagem e o redirecionamento para a mesma página, o que permitirá que a mensagem seja enviada mais de uma vez.

Infelizmente nesse momento não será possível testar o envio, pois precisamos estar com a aplicação publicada em um servidor para que as mensagens sejam encaminhadas ao servidor de destino. Mas não fique triste, na próxima parte do artigo faremos esse teste.

Conclusão

Nesse artigo aprendeu-se a utilizar *Templates* e como isso pode tornar-se benéfico para sua equipe de desenvolvimento. Também, montou-se todas as funcionalidades para nosso *Front End*.

Na próxima edição da revista teremos o último artigo desse mini-curso, onde faremos o *Deploy* da aplicação e a publicação em um servidor *web* válido de um provedor de hospedagem podendo fazer todos os testes, correções e melhorias na aplicação. Não perca a última parte! ●

DevMedia
GROUP

O conteúdo ao seu alcance!

+ de **600 artigos** impressos por ano

+ de **1500 vídeo-aulas** de acesso imediato

+ de **24 cursos** online

+ de **130 vídeos inéditos** todos os meses

Se você busca **conteúdo** de qualidade e aprendizagem rápida

Aqui é seu lugar!

Acesse e confira:

www.devmedia.com.br

Nesta seção você encontra artigos sobre a linguagem PHP e a ferramenta Delphi for PHP

Introdução ao PHP – Parte 3

Criação de funções, tratamento de exceções e uso de arquivos

Ewald Geschwinde
Hans-Juergen Schoenig

Funções são a chave para reusabilidade em qualquer linguagem de programação. Com PHP não é diferente, pois através de funções podemos centralizar nossas principais rotinas, tratamentos, regras de negócio etc. Nós já vimos na edição 89 como fazer uso de funções do próprio PHP, porém, até agora não implementamos nossas próprias funções. Nesse artigo aprenderemos a construí-las em PHP.

Funções Simples

Comecemos com um exemplo simples: a *Fórmula de Gauss* para a soma de números inteiros. Com sete anos de idade, Gauss iniciou a escola elementar, e o seu potencial foi notado quase imediatamente. O seu professor, Büttner, e o seu assistente, Martin Bartels, ficaram impressionados quando Gauss somou os números inteiros de 1 até 100, imediatamente deduzindo que a soma é formada por 50 pares de números, cada par somando 101. Assim nasceu a Fórmula de Gauss para a soma de números inteiros (**Listagem 1).**

Na primeira linha vemos a função de *Gauss* sendo chamada. Um parâmetro deve ser passado à função. Este parâmetro definirá o limite superior da soma que desejamos computar. Se o parâmetro passado para a função for um valor inteiro e se for maior do que o zero, o resultado é computado e retornado para a função principal. O resultado é passado para a função principal utilizando o comando *return*. Neste caso será retornado o valor 10 e a seguir, será exibido na tela o texto:

```
Soma de 1 até 4: 10
```

Podemos também chamar funções passando mais de um parâmetro. O exemplo da **Listagem 2** mostra uma versão ligeiramente adaptada da função que vimos anteriormente. Com a ajuda desta função, é possível computar a soma de valores inteiros, iniciando de um valor inicial e atingindo um valor final maior (o valor

inicial não está incluído na soma). Se executarmos o script, o resultado exibido na tela será:

```
Resultado: 45
```

Passando arrays para uma função

Às vezes é necessário passar *arrays* inteiros para uma função. O PHP não suporta a sobrecarga (*overload*) de função, como em Delphi, por exemplo, mas com a ajuda de *arrays*, é possível construir soluções de contorno bem simples. O objetivo do exemplo da **Listagem 3**, é computar a média geométrica de um *array* de valores:

Em primeiro lugar, é criado um *array* que contém vários valores. A seguir, a função *geomean* é chamada e o *array* é passado para a mesma. A função *geomean* computa a soma das segundas potências dos valores no *array*. Depois disso, a raiz quadrada é computada e retornada para a função principal. O resultado será exibido na tela:

```
Resultado: 3.8729833462074
```

Retornando mais de um Valor

Até agora, lidamos com funções que retornam apenas um valor. Contudo, podemos ter funções que retornam vários valores. Vejamos no exemplo na **Listagem 4**.

Em primeiro lugar, um *array* é gerado e passado para a função chamada *compute*, a qual executa algumas operações simples e retorna um *array* de valores. A saída é exibida na tela:

```
valor: 5
valor: 20
valor: 400
valor: 9600
valor: 153600
```

Como podemos ver, não faz nenhuma diferença se quisermos retornar uma variável comum ou um *array* de valores, a sintaxe é a mesma.

Tratando exceções

O tratamento de exceções é uma característica fundamental de quase toda linguagem de programação. O tratamento de exceções não é necessário somente para descobrir erros, mas também para o controle de fluxo. O PHP fornece um modelo poderoso para tratamento de exceções, que pode ser facilmente utilizado por programadores. Para mostrarmos como

Listagem 1. Fórmula de Gauss

```
<?php
  $result = gauss(4);
  echo "Soma de 1 até 4: $result<br>\n";
  # função para calcular a soma de 1 para $upper
  function gauss($upper){
    if(is_int($upper) && ($upper > 0)){
      return($upper*($upper+1)/2);
    }
  }
?>
```

Listagem 2. Fórmula de Gauss com mais de um parâmetro

```
<?php
  $lower = 4;
  $upper = 10;

  $result = gauss($lower, $upper);
  echo "Resultado: $result<br>\n";

  function gauss($lower, $upper){
    if($upper >= $lower && $lower >= 0){
      return ($upper*($upper+1)/2) - ($lower*($lower+1)/2);
    }
  }
?>
```

Listagem 3. Exemplo de passagem de array para funções

```
<?php
  $values = array(4, 3, 19, 23, 15);
  $result = geomean($values);
  echo "Resultado: $result<br>\n";

  function geomean($invalues){
    foreach ($invalues as $val){
      $sum += $val*$val;
    }
    return sqrt($val);
  }
?>
```

Listagem 4. Retornado mais de um valor na função

```
<?php
  $values = array(4, 3, 19, 23, 15);
  $result = compute($values);
  foreach($result as $val){
    echo "valor: $val<br>\n";
  }

  function compute($invalues){
    $result = array();
    $sum = 1;
    foreach($invalues as $val){
      $sum += $val*$sum;
      array_push($result, $sum);
    }
    return $result;
  }
?>
```

Listagem 5. Exemplo simples de tratamento de exceções

```
<?php
  $a = "uma string";
  if(sqrt($a)){
    echo "Resultado: $b";
  }else{
    echo "A operação não pode ser executada.";
  }
?>
```

Listagem 6. Exemplo de tratamento apenas com If

```
<?php
  $a = 3;
  if($a < 5){
    echo '$a é menor que 5';
    exit;
  }
  echo "Isso não pode ser executado.";
?>
```

realizar o tratamento de exceções, incluímos um exemplo simples. **Listagem 5**

Em primeiro lugar, é definida uma *string*. Então, o script tenta computar a raiz quadrada da *string*, o que não será possível. Por isso, a condição avaliada pela cláusula *if* será *false* e o ramo da cláusula *else* será executado.

Se qualquer coisa não permitida acontecer durante a execução do script, é necessário interromper a execução do programa. Isto pode facilmente ser realizado com a ajuda da função *die* (**Listagem 6**).

Como podemos ver, a variável *$a (3)* é menor do que 5, portanto a condição é verdadeira e o script pára mostrando a mensagem a seguir:

```
$a é menor que 5
```

A saída do comando *echo* no fim do script não será mais exibida. O comando *exit* não é o único fornecido pelo PHP para interromper um script. Se uma mensagem tiver que ser exibida antes de interromper um script, o comando *die*, como mencionado anteriormente, pode ser utilizado em lugar do comando *exit* como podemos ver a seguir:

```
<?php
  $a = 3;
  array_push($a, 4) or die ('$a não é um array');
  echo "Isso não será executado";
?>
```

Uma mensagem de erro PHP é exibida na tela. Além do mais, a mensagem que foi passada para o comando *die* também será exibida.

```
$a não é um array
```

Para aplicações do mundo real, a mensagem incorreta gerada pelo PHP não será conveniente, pois faz com que o usuário pense que ocorreu um erro na aplicação. Para garantir que nenhum erro seja exibido pelo PHP, podemos utilizar o seguinte:

```
<?php
  $a = 3;
  @array_push($a, 4) or die ('$a não é um
array');
  echo "Isso não pode ser executado.";
?>
```

Se executarmos o script, veremos que nenhum erro PHP gerado pela linguagem é mostrado, isso porque incluímos o caractere arroba (@) antes da função *array_push()* do PHP fazendo com que apenas o erro especificado por nós seja exibido:

```
$a não é um array
```

Como podemos ver, o PHP fornece um sistema flexível para tratamento de exceções, que pode ser facilmente utilizado para interceptar quase todos os erros que ocorrem durante a execução de um *script*.

Trabalhando com arquivos

Trabalhar com arquivos em PHP é fácil. Os arquivos não têm que estar localizados na máquina local para serem acessíveis. É possível também trabalhar com arquivos remotos. Nesta seção, aprenderemos a trabalhar eficientemente com arquivos locais e remotos.

Realizando operações básicas com arquivos locais

O trabalho com arquivos é um assunto importante em toda linguagem de programação. O mesmo aplica-se ao PHP. As funções para acessar o sistema de arquivos, permitem que o desenvolvedor construa aplicações altamente sofisticadas. Com a ajuda de arquivos, é possível armazenar informações a respeito do que está acontecendo em nossa aplicação ou, então, podemos acessar fontes de dados externas. Atualmente, muita informação é fornecida em bancos de dados, mas os arquivos são ainda um componente fundamental para as aplicações. Esta seção irá guiar-nos pelas funções do PHP relacionadas a arquivos e sistemas de arquivos. A primeira coisa que devemos saber a respeito de arquivos, é como abri-los e fechá-los como podemos ver a seguir:

```
<?php
 $handle = @fopen("data.file", "r") or
  die ("Não foi possível abrir o arquivo.");
  echo 'Arquivo aberto com sucesso.<br>';
  fclose($handle);
?>
```

No exemplo anterior são utilizadas as funções *fopen* e *fclose*. Estas funções estão disponíveis em muitas linguagens de programação e foram emprestadas do C. No script, um arquivo chamado *data.file* é aberto para leitura. Isto é feito com a função *fopen*. O primeiro parâmetro define o nome do arquivo que deve ser aberto. O segundo parâmetro diz ao PHP em que modo o arquivo deve ser aberto. O PHP suporta os seguintes modos:

- r – Leitura somente;
- r+ - Abre para leitura e gravação;
- w – Abre o arquivo para gravação e cria um arquivo vazio;
- w+ - Abre o arquivo para leitura e gravação;
- a - Abre o arquivo para gravação e posiciona o ponteiro no final do mesmo;
- a+ - Abre o arquivo para leitura e gravação e posiciona o ponteiro no final do mesmo;

A função *fclose* fecha o arquivo. Vejamos o conteúdo do arquivo que o script processou e imprimiu em tela:

```
Guinther::Brasil::Editor Geral
Adriano::Brasil::Editor Técnico
Gladstone::Brasil::Diretor
João Morais::Brasil::Leitor
```

Às vezes é necessário descobrir alguma coisa a mais sobre um arquivo. Por isso, o PHP provê um comando chamado *stat*. A **Listagem 7** mostra o uso da função mencionada.

Ao executarmos o script, muita informação é exibida em tela como mostrado a seguir:

```
dispositivo: 773
inode (nó): 470215
inode (nó) de modo de proteção: 33188
número de links: 1
id do usuário proprietário: 500
grupo do usuário proprietário: 500
tipo de dispositivo: 2817
tamanho em bytes: 107
último acesso: 1003663763
última modificação: 1003663300
última mudança: 1003663300
tamanho do bloco de sistema de arquivos: 4096
número de blocos alocados: 8
```

A saída exibida pelo comando *stat* contém a mesma informação que é retornada pela função *stat* do C na qual o PHP se baseou. Para demonstrar que as funções do PHP para trabalhar com sistemas de arquivos, foram baseadas nas funções correspondentes do C, incluímos a estrutura retornada pela função *stat* do C na **Listagem 8**.

Como podemos ver, o conteúdo do resultado gerado pelo PHP é quase igual ao resultado gerado pelo C. Isto mostra claramente que o PHP e o C estão fortemente relacionados. O conhecimento disto facilitará o entendimento do comportamento e das funções do interpretador do PHP.

Lendo e escrevendo em arquivos

Até agora, aprendemos a abrir e fechar arquivos. Também vimos como descobrir informações a respeito de um arquivo, mas ainda não lemos nem escrevemos dados. Comecemos com um exemplo onde podemos ver como os dados são lidos em um arquivo (**Listagem 9).**

Em primeiro lugar, um arquivo é aberto e a informação do mesmo é recuperada utilizando o comando *stat*. A variável *$fileinfo[7]* conterá o tamanho do arquivo. Embora tenhamos passado o conteúdo de *$fileinfo[7]* para a função *fgets*, só a primeira linha será lida, pois a função *fgets* interpreta caracteres *newline*("nova linha"), ou seja, depois de ler a primeira linha, o ponteiro é posicionado no início da segunda linha.

Então a função *fread* é utilizada. Desta vez, o resto do arquivo será lido e exibido na tela, pois a função *fread* não processa os caracteres *newline*. Depois de ler todos os dados, o ponteiro do arquivo é posicionado no fim. Para executar novas leituras, é necessário posicioná-lo em uma posição diferente. Para isto, a função *rewind* é utilizada para posicionar o ponteiro na primeira posição do arquivo. Depois disso, a função *fseek* posiciona o ponteiro na 20ª posição no arquivo. As funções *fseek* e *rewind*, são duas funções essenciais para trabalhar com arquivos, pois podemos usá-las para posicionar o ponteiro em qualquer ponto do arquivo com o qual estamos trabalhando. Vejamos a saída do script:

```
Leitura usando fgets:
Hans::Vienna::Database Developer

Leitura usando fread:
Epi::Vienna::Consultant Kuli::Vienna::Sales
Manager Sunny::Murau::Student

O ponteiro está na posição: 0
O ponteiro está na posição: 20
```

Como podemos ver, o resultado é exibido na tela corretamente. Já na **Listagem 10**, podemos ver como os dados são escritos em um arquivo:

A variável *$handle* contém o ponteiro do novo arquivo criado pela função *fopen*. Então, a função *fputs* é chamada e uma string é gravada no arquivo antes do seu fechamento.

Até agora, só utilizamos funções de entrada/saída rudimentares. Em alguns casos, é mais cômodo ler um arquivo inteiro e colocá-lo em um *array*. Observe a **Listagem 11**.

A leitura de um arquivo para um *array* pode ser feita com uma função chamada *file*. Para utilizar o arquivo, não será necessário utilizar primeiramente a função *fopen* para abrir o arquivo, pois isto é feito implicitamente e um *array* é retornado. No

Listagem 7. Uso da função stat

```
<?php
  $fileinfo = stat("data.file") or
    die ("Não foi possível encontrar informações sobre o arquivo.");
  echo "dispositivo: $fileinfo[0]<br>\n";
  echo "inode (nó): $fileinfo[1]<br>\n";
  echo "inode (nó) do modo de proteção: $fileinfo[2]<br>\n";
  echo "número de links: $fileinfo[3]<br>\n";
  echo "id do usuário proprietário: $fileinfo[4]<br>\n";
  echo "grupo do usuário proprietário: $fileinfo[5]<br>\n";
  echo "tipo de dispositivo: $fileinfo[6]<br>\n";
  echo "tamanho em bytes: $fileinfo[7]<br>\n";
  echo "último acesso: $fileinfo[8]<br>\n";
  echo "última modificação: $fileinfo[9]<br>\n";
  echo "última mudança: $fileinfo[10]<br>\n";
  echo "tamanho do bloco do sistema de arquivos: $fileinfo[11]<br>\n";
  echo "número de blocos alocados: $fileinfo[12]<br>\n";
?>
```

Listagem 8. Estrutura da função stat do C

```
struct stat {
  dev_t st_dev;             /* dispositivo */
  ino_t st_ino;             /* nó */
  mode_t st_mode;           /* proteção */
  nlink_t st_nlink;         /* número de links */
  uid_t st_uid;             /* ID do usuário proprietário */
  gid_t st_gid;             /* ID do grupo proprietário */
  dev_t st_rdev;            /* tipo de dispositivo */
  off_t st_size;            /* tamanho em bytes */
  unsigned long st_blksize; /* tamanho do bloco de sistema de arquivos*/
  unsigned long st_blocks;  /* número de blocos alocados */
  time_t st_atime;          /* último acesso */
  time_t st_mtime;          /* última modificação */
  time_t st_ctime;          /* última mudança */
};
```

Listagem 9. Exemplo de leitura de arquivos

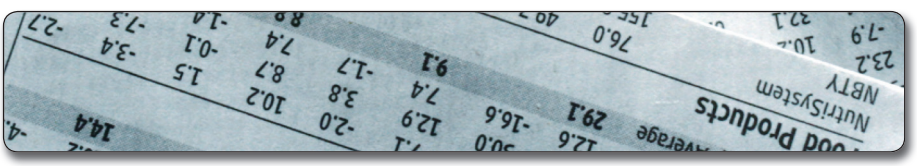
```
<?php
  $file = "data.file";
  $handle = fopen($file, "r") or
    die ("Não é possível abrir o arquivo.");
  $fileinfo = stat($file);
  $data = fgets($handle, $fileinfo[7]);
  echo "Leitura usando fgets: <br>$data<br><br>";
  $data = fread($handle, $fileinfo[7]);
  echo "Leitura usando fread: <br>$data<br><br>";
  rewind($handle);
  echo "O ponteiro está na posição: ".ftell($handle)."<br>";
  fseek($handle, 20);
  echo "O ponteiro está na posição: ".ftell($handle);
  fclose($handle);
?>
```

Listagem 10. Escrita do arquivo

```
<?php
  $file = "data.file2";
  $handle = fopen($file, "w") or
    die ("Não é possível abrir o arquivo.");
  fputs($handle, "Olá Mundo!");
  fclose($handle);
?>
```

Listagem 11. Exemplo de leitura de arquivo para array

```
<?php
  $file = "data.file";
  $data = file($file);
  echo '<table border="1"><tr>';
  echo "<th>nome</th><th>localização</th><th>profissão" . "</th></tr>";
  foreach($data as $line){
    $val = explode("::", $line);
    echo "<tr><td>$val[0]</td><td>$val[1]</td><td>
      $val[2]</td></tr>";
    }
  echo "</table>";
?>
```



exemplo, o *array* é processado linha a linha e a string na linha é dividida após cada conjunto de caracteres "::". Desta maneira, um *array* chamado *$val* é inicializado sempre que a repetição é processada.

Se quisermos exibir o conteúdo do arquivo na tela, sem processar previamente os dados, podemos utilizar a função *readfile*. A função *readfile(nome_do_arquivo)* abre um arquivo e exibe o conteúdo diretamente na tela. Isto é muito útil quando o arquivo já contém o código HTML.

Realizando operações básicas com arquivos remotos

Na seção anterior, vimos como os arquivos locais são tratados com PHP. Os arquivos remotos são processados de um modo semelhante. Observemos o exemplo que acabamos de ver, mas desta vez os dados vêm de um serviço de hospedagem, ou seja, de um servidor remoto. Na **Listagem 12** podemos ver que há poucas mudanças na leitura do arquivo em relação ao exemplo da seção anterior.

Como podemos verificar, a única modificação que deve ser feita, consiste em que a posição do arquivo é definida agora por uma *URL*. Se tivermos as permissões para ler o arquivo, este poderá ser processado como qualquer outro arquivo. Sempre que o *script* é processado, contudo, o servidor *Web* acrescentará uma linha ao *logfile* (arquivo de *log*):

```
212.186.25.254 - - [21/Oct/2001:16:30:02
+0200] "GET /test/data.file HTTP/1.0" 200 107
"-" "PHP/4.0.4pl1"
```

Listagem 12. Leitura do arquivo remotamente

```
<?php
  $file = "http://212.186.25.254:/test/data.file";
  $data = file($file);
  echo '<table border="1"><tr>';
  echo "<th>name</th><th>location</th><th>profession</th></tr>";
  foreach($data as $line){
    $val = explode("::", $line);
    echo "<tr><td>$val[0]</td><td>$val[1]</td><td>
    $val[2]</td></tr>";
  }
  echo "</table>";
?>
```

Listagem 13. Leitura remota do arquivo

```
<?php
  $file = "http://212.186.25.254:/test/data.file";
  $handle = fopen($file, "a+") or
    die ("Não é possível ler o arquivo $file");
      fputs($handle, "Patrick::Brasil::Presidente") or
    die ("Não é possível escrever no arquivo");
      fclose($handle) or
    die ("Não é possível fechar o arquivo.");
  echo "Funcionou";
?>
```

Como já mencionamos, o trabalho com arquivos remotos funciona da mesma forma como com arquivos locais. Contudo, alguns aspectos importantes têm de ser levados em consideração (**Listagem 13**).

Imaginamos que o *script* acrescentará os dados ao arquivo remoto, mas isto não acontece:

```
[guinther@devmedia.com.br teste]$ categoria
data.file
Guinther::Brasil::Editor Geral
Adriano::Brasil::Editor Técnico
Gladstone::Brasil::Diretor
João Morais::Brasil::Leitor
```

Nada foi acrescentado ao arquivo remoto, embora a saída do script tenha sido "funcionou" e nenhum erro de execução tenha ocorrido. Isso deve ser levado em consideração ao trabalharmos com arquivos remotos, caso contrário ficaríamos intrigados pelo fato da aplicação não retornar nenhum erro e apesar disso, não produzir nenhuma saída.

Funções adicionais do sistema de arquivos

Como já vimos, o PHP oferece funções para trabalhar com arquivos locais e remotos. Até agora, vimos como executar todas as operações básicas. Nesta seção, veremos que o PHP fornece muito mais funções do que as já vistas:

• *string basename (string path [, string suffix])* – Se passarmos um nome de arquivo incluindo o caminho (*path*) para a função *basename*, será retornado o nome do arquivo sem o caminho;

• *int chgrp (string filename, mixed group)* - *chgrp* pode ser utilizada para alterar o grupo ao qual um arquivo pertence;

• *int chmod (string filename, int mode)* - *chmod* pode ser utilizada para alterar o modo de um arquivo. Esta função não funciona com servidores Windows;

• *int chown (string filename, mixed user)* - *chown* pode ser utilizada para alterar o proprietário de um arquivo;

• *void clearstatcache (void)* – limpa o *cache* de estado do arquivo (*file stat cache*). Normalmente o resultado de *stat* é armazenado em *cache* para economizar o tempo necessário para recuperar as informações de arquivos. Com o auxílio do *clearstatcache*, podemos forçar o PHP a re-executar a função de verificação de status;

• *int copy (string source, string dest)* – Utilizada para copiar arquivos.

• *string dirname (string path)* – dirname é a contrapartida da função *basename*. Retorna o caminho de um arquivo;

• *float diskfreespace (string directory)* – Utilizada para obter o espaço vazio disponível no diretório atual;

• *float disk_total_space (string directory)* – Retorna a capacidade total de armazenamento disponível na partição do diretório passado para a função;

• *bool fclose (int fp)* – Fecha um arquivo aberto;

• *int feof (int fp)* – Verifica se o ponteiro está posicionado no *EOF* (*end of file* – fim do arquivo);

• *int fflush (int fp)* – Descarrega em disco o conteúdo do *buffer*;

• *string fgetc (int fp)* – Lê um único caractere do arquivo;

• *array fgetcsv (int fp, int length [,string delimiter])* – Lê valores separados por vírgula (*CSV – comma separated values* – valores separados por ponto e vírgula) e retorna um *array* de valores. O delimitador de linha pode opcionalmente ser passado para a função;

• *string fgets (int fp, int length)* – Utilizado para ler do arquivo;

• *string fgetss (int fp, int length [,string allowable_tags])* – Equivalente ao *fgets*, porém, tenta extrair *tags* HTML;

• *array file (string filename [, int use_include_path])* – Retorna o conteúdo de um arquivo em um *array*;

• *bool file_exists (string filename)* – Retorna *True* caso o arquivo passado exista;

Existem coisas que não conseguimos ficar sem!

...só pra lembrar, sua assinatura está acabando!

Renove Já!

www.devmedia.com.br/renovacao

• *int fileatime (string filename)* – Recupera a data e hora de acesso de um arquivo;

• *int filectime (string filename)* – Retorna a data e hora de modificação do número inode (nó);

• *int filegroup (string filename)* - Retorna o grupo ao qual o arquivo pertence;

• *int fileinode (string filename)* - Recupera o número inode do arquivo;

• *int filemtime (string filename)* - Retorna a data e hora da modificação;

• *int fileowner (string filename)* - Retorna o proprietário do arquivo;

• *int fileperms (string filename)* - Retorna as permissões para o arquivo;

• *int filesize (string filename)* - Retorna o tamanho do arquivo;

• *string filetype (string filename)* - Retorna o tipo do arquivo (*fifo, char, dir, block, link, file ou unknown*;.

• *bool flock (int fp, int operation [, int wouldblock])* – Utilizado para bloquear arquivos;

• *int fopen (string filename, string mode [, int use_include_path])* – Abre um arquivo ou uma URL e retorna um handle para o arquivo;

• *int fpassthru (int fp)* – Escreve todos os dados do arquivo a partir da posição atual do ponteiro até o final do arquivo na saída padrão;

• *int fputs (int fp, string str [, int length])* – Escreve os dados para um arquivo, iniciando pela posição atual do ponteiro do arquivo;

• *string fread (int fp, int length)* – Leitura de arquivo binário-segura (Binary-safe);

• *mixed fscanf (int handle, string format [,string var1...])* – Interpreta a entrada com base em um formato;

• *int fseek (int fp, int offset [, int whence])* – Utilizado para mover o ponteiro do arquivo dentro do mesmo. Pode ser: *SEEK_SET* (posicionar o ponteiro do arquivo em *offset bytes* de deslocamento a partir do início); *SEEK_CUR* (posicionar o ponteiro do arquivo em *offset bytes* de deslocamento a partir da posição atual do mesmo); *SEEK_END* (posicionar o ponteiro do arquivo para fim do arquivo mais *offset bytes* de deslocamento);

• *array fstat (int fp)* – Recupera informações sobre o arquivo;

• *int ftell (int fp)* – Utilizado para descobrir em que posição o ponteiro de arquivo pode ser encontrado;

• *int ftruncate (int fp, int size)* – Utilizado para truncar o arquivo para um dado comprimento;

• *int fwrite (int fp, string string, int [length])* – Escrita no arquivo binário-segura (Binary-safe);

• *int set_file_buffer (int fp, int buffer)* – Modifica o buffer de arquivo para um dado tamanho. Por padrão, este tamanho é de 8 *Kbytes*;

• *bool is_dir (string filename)* – Utilizado para descobrir se filename é um diretório;

• *bool is_executable (string filename)* – Verifica se um dado arquivo é executável;

• *bool is_file (string filename)* – Verifica se filename é um arquivo;

• *bool is_link (string filename)* – Verifica se filename é um link simbólico;

• *bool is_readable (string filename)* - Verifica se o arquivo pode ser lido;

• *bool is_writable (string filename)* e *bool is_writeable (string filename)* - Verifica se o arquivo pode ser gravado;

• *bool is_uploaded_file (string filename)* – Retorna *True* se o arquivo foi carregado utilizando *HTTP POST*;

• *int link (string target, string link)* – Cria um link;

• *int linkinfo (string path)* – Retorna informações sobre um link;

• *int mkdir (string pathname, int mode)* – Cria um novo diretório;

• *bool move_uploaded_file (string filename, string destination)* – Utilizado para mudar a localização de um arquivo carregado;

• *array pathinfo (string path)* - Retorna informações sobre o caminho em um *array*;

• *int pclose (int fp)* – Fecha um *pipe* aberto por um *popen*;

• *int popen (string command, string mode)* – Abre um *pipe*;

• *int readfile (string filename, int [use_include_path])* – Lê dados a partir de um arquivo e imprime o conteúdo na saída padrão;

• *int rename (string oldname, string newname)* – Renomeia um arquivo;

• *int rewind (int fp)* – Posiciona o ponteiro no início do arquivo;

• *int rmdir (seqüência dirname)* – Remove o diretório;

• *array stat (string filename)* – Recupera informações detalhadas sobre o arquivo;

• *array lstat (string filename)* – Retorna informações sobre um link;

• *int symlink (string target, string link)* – Gera um link simbólico para um arquivo;

• *string tempnam (string dir, string prefix)* – Cria um *filename* temporário único;

• *int tmpfile (void)* – Cria um arquivo temporário com um filename único;

• *int touch (string filename, int [time])* – Ajusta a data e hora da modificação do arquivo para a data e hora atual;

• *int unlink (string filename)* – Apaga um arquivo;

Como podemos ver, o PHP fornece uma lista enorme de funções embutidas. A maior parte das funções funciona com todos os sistemas de arquivos comuns. Principalmente em sistemas Unix, o PHP demonstra um tremendo poder, pois algumas funções só são utilizadas em Unix.

Conclusão

Nessa terceira parte de introdução ao PHP vimos como é importante saber trabalhar com funções de modo que podemos armazenar nossas principais regras de negócios em rotinas centralizadas. Dessa forma fica muito mais simples fazer a manutenção do sistema e desenvolver novas soluções sempre que necessário.

Vimos também como fazer o tratamento de exceções, premissa básica de qualquer linguagem de programação. Por últimos aprendemos como trabalhar e quais a diferenças entre utilizar arquivos localmente e remotamente. ●

A EDIÇÃO QUE VOCÊ PRECISA
ESTÁ ESGOTADA?

SEUS PROBLEMAS ACABARAM!!!

Seja um assinante Gold!

Com a assinatura Gold você já pode consultar online
todos os artigos publicados na sua revista desde a edição nº 1.

DevMedia GROUP

Saiba Mais! Acesse:
www.devmedia.com.br/assgold

Para mais informações:
www.devmedia.com.br/central

Assinatura Gold

Atenção: Já encontram-se disponíveis as assinaturas GOLD das revistas WebMobile e .net Magazine

NOVOS PLANOS

Descubra porque Hospedix é a hospedagem preferida dos profissionais.

Plano **Profissional** 1

1 Registro de domínio grátis*
3 Domínios por conta
50 GB de transferência
30 GB para e-mails com SSL
1 GB de espaço
ASP, ASP.NET 2.0 e PHP 5
Access e MySQL 5
SQL Server 2005 Express
Painel de controle e webmail
Editor de páginas on-line
Anti-spam e Anti-vírus

R$ 25,90 por mês

30 DIAS GRÁTIS
PARA EXPERIMENTAR
Ao assinar, digite o código de desconto abaixo exclusivo para leitores da .NET Magazine e ganhe o primeiro mês inteiramente grátis.

RVSTNET

DOMÍNIO GRÁTIS
ENQUANTO FOR CLIENTE
Só na Hospedix você tem registro de domínio (.com .net .org ou .info) grátis e isento de taxas anuais enquanto for cliente.

PAINEL DE CONTROLE
PLESK 8.1 PROFISSIONAL
Painel de controle completo onde você mesmo adiciona domínios, sub-domínios, cria contas de e-mail, e muito mais.

Transfira seu site hoje mesmo para a Hospedix e descubra porque somos a empresa de hospedagem preferida dos profissionais.

www.hospedix.com.br

HOSPEDIX
HOSPEDAGEM PROFISSIONAL